

21世纪大学本科计算机专业系列教材

蔡晓燕　编著

FPGA数字逻辑设计

http://www.tup.com.cn

- 根据教育部“高等学校计算机科学与技术专业规范”组织编写
- 与美国ACM和IEEE CS *Computing Curricula* 最新进展同步

清华大学出版社

21 世纪大学本科计算机专业系列教材

FPGA 数字逻辑设计

蔡晓燕　编著

清华大学出版社
北　京

内 容 简 介

本书是为"数字逻辑电路"等课程配套的实验教材。作为专业基础课程的配套实验，其主要目的是为学生学习后续硬件类课程培养硬件设计基础和实验技能。本书首先介绍了可编程器件、数字系统设计方法、电子设计自动化软件、Verilog HDL 程序设计方法等基础知识。在此基础上设计了 18 个实验题目，从组合逻辑电路设计、时序逻辑电路到状态机设计以及常用接口控制器设计。每个实验都从理论知识入手，先给出引导性实验，再进入设计性实验，知识的介绍和实验的要求循序渐进、由浅入深，不仅逻辑严密，而且操作性强。

本书适合作为高等学校计算机类专业及相关专业"数字逻辑"等课程的实验教材，也可供其他领域从事数字系统设计的工程技术人员参考。

图书在版编目（CIP）数据

FPGA 数字逻辑设计/蔡晓燕编著. --北京：清华大学出版社，2013.5
21 世纪大学本科计算机专业系列教材
ISBN 978-7-302-30975-8

Ⅰ. ①F…　Ⅱ. ①蔡…　Ⅲ. ①硬件描述语言－数字电路－计算机辅助设计－高等学校－教材
Ⅳ. ①TN790.2

中国版本图书馆 CIP 数据核字(2012)第 301685 号

责任编辑： 张瑞庆
封面设计：
责任校对： 李建庄
责任印制：

出版发行： 清华大学出版社
网　　址： http://www.tup.com.cn，http://www.wqbook.com
地　　址： 北京清华大学学研大厦 A 座　　**邮　　编：** 100084
社 总 机： 010-62770175　　**邮　　购：** 010-62786544
投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载： http://www.tup.com.cn，010-62795954
印 刷 者：
装 订 者：
经　　销： 全国新华书店
开　　本： 185mm×260mm　　**印　　张：** 11　　**字　　数：** 273 千字
版　　次： 2013 年 5 月第 1 版　　**印　　次：** 2013 年 5 月第1 次印刷
印　　数： 1～　000
定　　价： 　.00 元

产品编号：048122-01

前　言

FOREWORD

在过去的几十年中，集成电路的发展给科学技术和社会生活带来了前所未有的巨大改变，在一块 FPGA 上实现一个高性能的 CPU 已经能够实现。计算机辅助设计技术的发展，给数字电路设计带来了革命性的发展，将计算机辅助设计技术融入硬件设计中，使原来复杂的数字系统设计变得更加简单，这使得非电子工程专业的技术人员也能参与到硬件设计工作中。另一方面，随着嵌入式应用的发展和对计算机系统性能要求的提高，数字系统的软硬件协同设计也越来越重要，这也要求有计算机系统结构知识的人参与到硬件设计中来。

现在的工程师大多是通过硬件描述语言来设计数字系统的，常用的硬件描述语言是 VHDL 和 Verilog HDL。这两种语言没有优劣之分，应用都很广泛，本书采用 Verilog 硬件描述语言，大部分的实验例题都是采用电路行为级的描述，而不是用传统的电路逻辑方程的方式来描述电路。

目前国内的 FPGA 市场主要由 Xilinx 公司和 Altera 公司两大生产厂商占领，每个公司都为自己的 FPGA 开发了编译器。本书的所有例题都是在 Altera 公司的 FPGA 上进行实验的，因此本书中例题的编译平台是 Altera 公司的 Quartus Ⅱ。如果读者使用的是 Xilinx 公司的开发平台，只要将本书中实例的 Verilog HDL 代码输入至 Xilinx ISE 编译器中重新编译，即可产生可下载到 Xilinx 公司的 FPGA 中的文件。

本书共 6 章。第 1 章简单介绍数字逻辑芯片。第 2 章简单介绍 EDA 技术的基础知识，并且通过两个实例让读者对 Quartus Ⅱ的使用和用 Verilog HDL 来设计硬件电路有简单的了解。第 3 章和第 4 章分别介绍组合逻辑电路和时序逻辑电路的设计方法。第 5 章和第 6 章分别介绍简单数字系统设计和常用的 I/O 接口的相关知识。附录简要介绍竞争、冒险和毛刺现象，并介绍了消除毛刺的方法。本书的编写采取由简到繁、循序渐进逐渐加深的方法，对每个内容都是先给出具体的实例，让读者初步了解相关知识，然后提高难度，引导读者独立思考设计出自己的相对复杂的数字电路。因此，只要有数字逻辑电路基本理论知识的读者都可以阅读本书，对于有想设计出更加复杂的数字系统的读者，阅读本书也会有所帮助。

由于目前国际上流行的大多数数字技术教材、数字系统设计资料和主流 EDA 软件中，一直流行采用 ANSI/IEEE 91—84 标准特定外形的图形符号，因此，本教材也主要使用这类二进制逻辑元件符号。

关于本书实验教学的建议如下。

1. 实验安排

本书共设计了 18 个实验，每个实验有若干个引导性实验和设计性实验项目，授课时可以根据专业和学生层次灵活选择。

实验课程安排为每周一个实验，课堂实验时间为 2～3 学时，课前需要对实验项目进行预习，根据书中提示完成引导性实验，设计性实验要求在实验课前完成电路或代码设计，电路或代码无语法错误。

2. 教学和考核

建议每次实验课前，教师根据课堂情况和实验报告情况对上一次实验进行总结，对普遍存在的问题统一讲解。尽量在每次实验课上介绍下一次实验的要点，要求学生对相关理论知识、工具使用等进行课前预习，完成预习报告，任课教师在实验开始前要对本次实验的预习报告进行检查或抽查。实验时可以对每个完成的实验进行验收、提问并作记录。课后在预习报告的基础上完成内容完整、条理清楚的实验报告。

学期实验完成时，可以单独设计实验进行考核，也可以每次实验都设计考核内容。成绩评定根据实验预习、课堂验收提问、实验报告和期末考核进行综合评分。

在本书的编写过程中得到了许多教师和学生的帮助。在此特别感谢袁春风教授、张泽生高级工程师和李宇鹏同学的帮助。

作　者

2013 年 3 月于南京大学

目　录

CONTENTS

第 1 章 逻辑器件简介

我们生活在一个数字化的世界中，小到电子手表、电子玩具，大到计算机、服务器等都是数字系统。数字逻辑器件（集成电路）是数字系统赖以工作的硬件基础，集成电路（IC）的发展水平决定着数字逻辑电路的复杂程度和工作性能。英特尔公司的创始人之一戈登·摩尔（Gordon Moore）指出：集成电路上可容纳的晶体管数目每隔 18 个月便会增加一倍，性能也将提升一倍，而且其价格保持不变，这就是有名的"摩尔定律"。目前，集成电路的工艺水平已经到了几十纳米级。

本章首先介绍几种简单的集成电路，然后介绍复杂可编程逻辑电路芯片（CPLD）的工作原理，最后详细描述 Altera 公司 Cyclone Ⅱ 系列现场可编程逻辑阵列（FPGA）的结构。

1.1 逻辑器件概述

随着集成电路工艺水平的发展，在数字逻辑器件的发展过程中，集成电路芯片经历了几个不同的发展阶段，本节简单介绍各个阶段出现的被普遍使用的几种芯片。

1.1.1 固定逻辑芯片

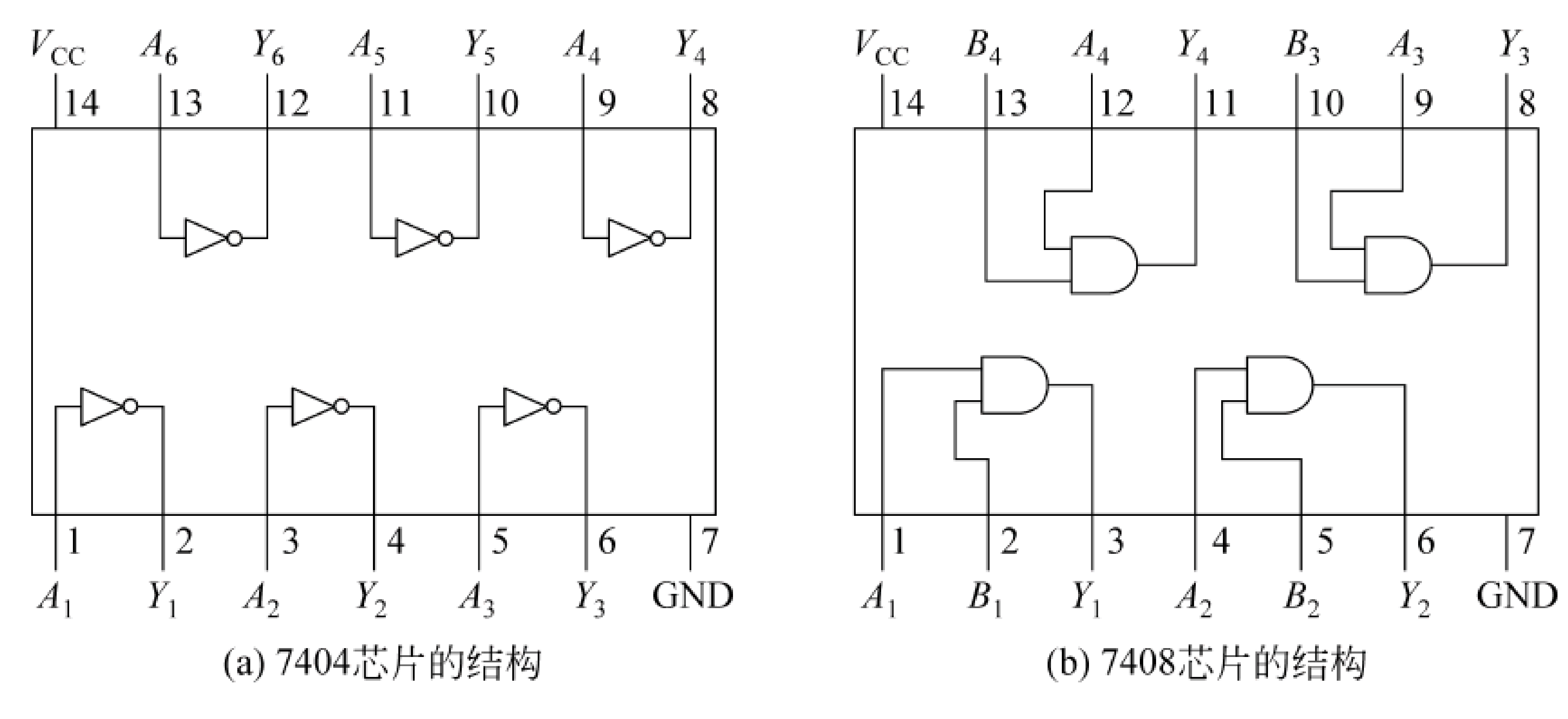
固定逻辑的标准芯片曾被广泛使用，了解其结构有助于理解可编程芯片的结构。因此，我们从简单的固定逻辑芯片开始介绍。

20 世纪 80 年代中期以前，数字逻辑电路设计中采用的芯片一般是标准芯片，例如 7400 系列芯片。7400 系列芯片的每个芯片中只含有少数几个功能相同的逻辑门，例如，7404 芯片中含有 6 个单输入非门，7408 芯片中含有 4 个双输入与门，如图 1-1 所示。

7400 系列包含许多不同的芯片，使用时需要查阅芯片制造商提供的数据手册。在稍为复杂的电路设计过程中，通常需要选择多个不同功能的芯片，并将其按照一定方式连接起来形成所需要的电路。

下面举例说明如何使用 7400 系列芯片实现一个具体的逻辑电路。

假设需要实现的电路的逻辑功能是 $F=\overline{AB}$，在这个逻辑函数中，需要完成一个二输入的与门和一个非门，因此，我们可以选用 74LS04 芯片和 74LS08 芯片来实现此逻辑，如



(a) 7404芯片的结构　(b) 7408芯片的结构

图 1-1　74LS04 和 74LS08 芯片结构图

图 1-2 所示(实际设计中可以选用现成的与非门芯片 74LS01,本例只是用来说明问题)。请注意,所有的芯片在使用时其电源(V_{CC})端都要连接到 5V 电源端,所有的地(GND)端都要连接到接地端。

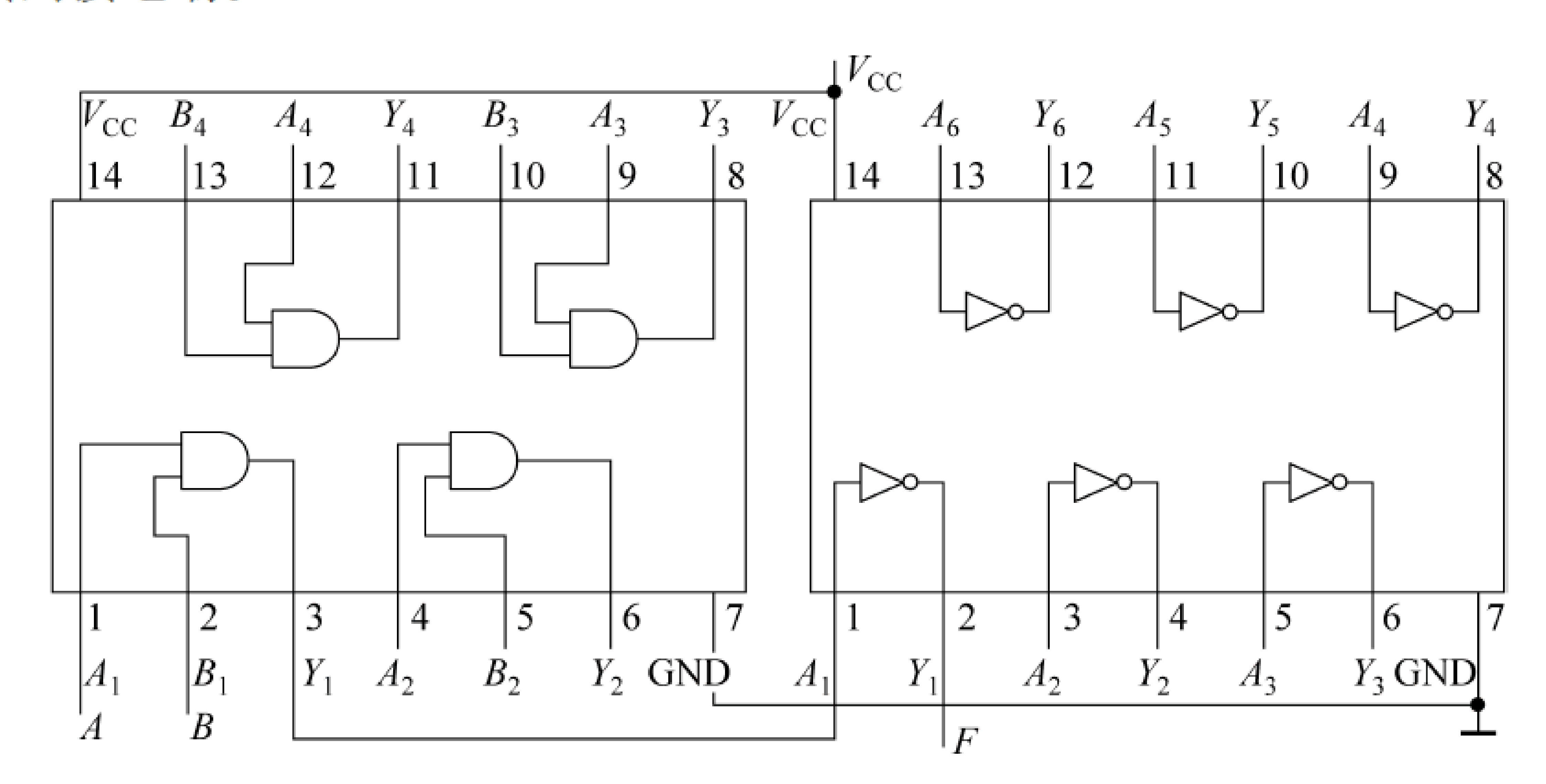


图 1-2　$F=\overline{AB}$的电路逻辑图

固定逻辑芯片属于中、小规模集成电路(SSI),一片芯片的逻辑门数在 100 门以下,且其逻辑功能单一、固定,不能随着电路设计的需求而任意改变其逻辑功能。随着电路复杂度的提高,利用中、小规模集成电路设计电路的难度将大大增加,电路稳定性也随之降低。

随着集成电路技术的迅速发展,在 20 世纪 80 年代后期出现了大规模可编程逻辑器件(Programmable Logic Device,PLD),可编程逻辑器件给逻辑电路的设计带来了前所未有的灵活性。

1.1.2　简单 PLD 器件

可编程逻辑器件(PLD)是相对于固定功能的逻辑器件而言的。PLD 是一种用于实现逻辑电路的通用器件,其中包含多个逻辑单元,可以根据客户的需要进行编程,构成不同功能的逻辑电路。PLD 的结构框图如图 1-3 所示,芯片内部含有多个逻辑门和编程开关,逻辑门可以通过编程开关连接起来,形成所需要的逻辑电路。

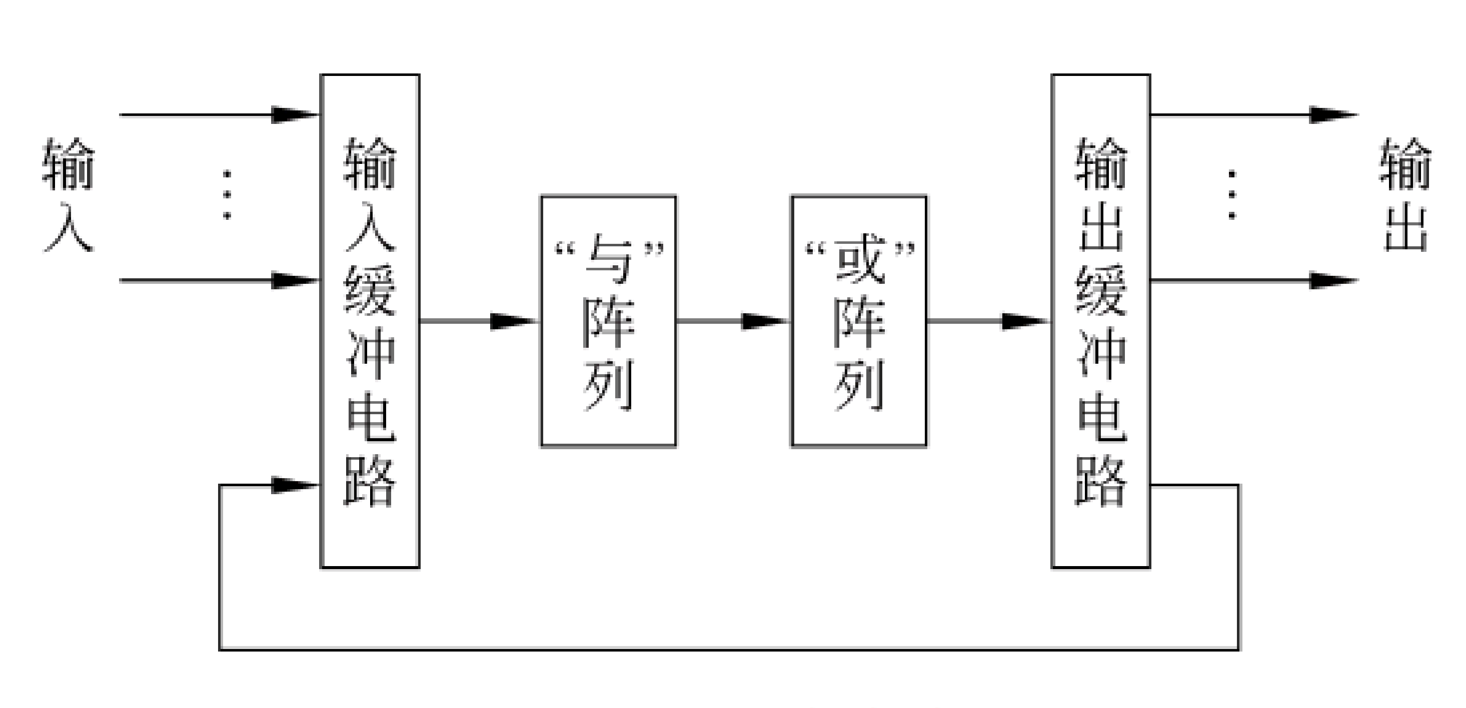


图 1-3　PLD 结构框图

1.1.2.1　PLD 中的电路符号表示

PLD 中常用的基本电路符号表示如图 1-4 所示。

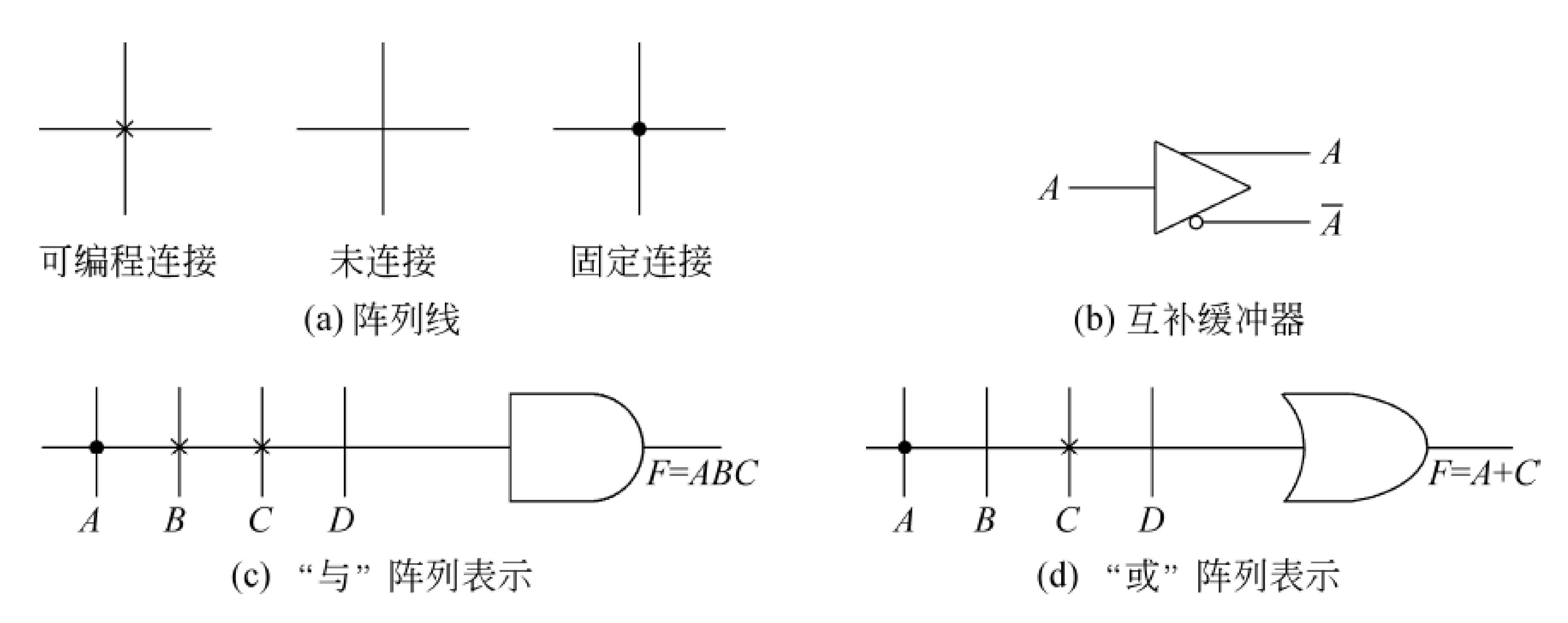


图 1-4　PLD 中常用的基本电路符号表示

图中连接线上的叉(×)表示 PLD 芯片中被编程为"连接"的可编程节点/开关,也可以将这些节点设置为"不连接"方式,通过对这些节点的适当配置或编程,就能实现用户需要的逻辑电路。使用 PLD 时,逻辑电路设计人员利用计算机辅助设计(Computer Aided Design,CAD)工具,在计算机上用电路原理图或者硬件描述语言(Hardware Description Language,HDL)描述出电路的功能,这些支持 PLD 器件的 CAD 工具(例如 Altera 公司的 MAX+PLUS Ⅱ和 Quartus Ⅱ)能够自动生成针对 PLD 器件中每一个开关的编程信息,并产生编程文件。然后,将运行 CAD 工具的计算机通过电缆线与 PLD 开发平台相连,将含有编程信息的文件传送给 PLD 开发平台,开发平台上的编程器可以根据编程文件通过对 PLD 进行编程完成对 PLD 的配置,从而实现电路的设计。

1.1.2.2　简单 PLD 器件的基本结构

PLD 的种类繁多,分类不一。如果按照集成度来划分,可分为两类:简单 PLD 和复杂 PLD。逻辑门数 500 门以下的被认为是简单 PLD,包括 PROM、PLA、PAL、GAL 等器件;而逻辑门数在 500 门以上且芯片集成度高的则被称为复杂 PLD,包括 EPLD、CPLD、FPGA 等器件。目前,逻辑电路设计中常用的则是 CPLD 和 FPGA 器件。

了解简单 PLD 的基本结构有助于深入理解复杂 PLD 的结构,下面分别介绍几种简单 PLD 的基本结构。

1. 简单 PROM

图 1-5 是一种简单 PROM 的结构图。PROM 是一种“与”阵列固定、“或”阵列可编程的简单 PLD,在图左侧的“与”阵列中,每个与门与一条水平横线连接,而作为与门输入的信号线画成垂直线,与水平线相交,水平线和垂直线的某些交点处在硬件上设置为固定连接,形成固定的与门逻辑。图的右侧为“或”阵列,每一个或门与一条垂直线连线,这些垂直线垂直相交于“与”阵列的输出线,选择需要的与门输出,在连线交点处打个叉(×)表明该输入被编程为和与门连接,以此来实现所需要的逻辑功能。图 1-5 实现的逻辑功能是:

$$F_1 = \bar{A}B + AB \qquad F_2 = \bar{A}\bar{B} + A\bar{B}$$

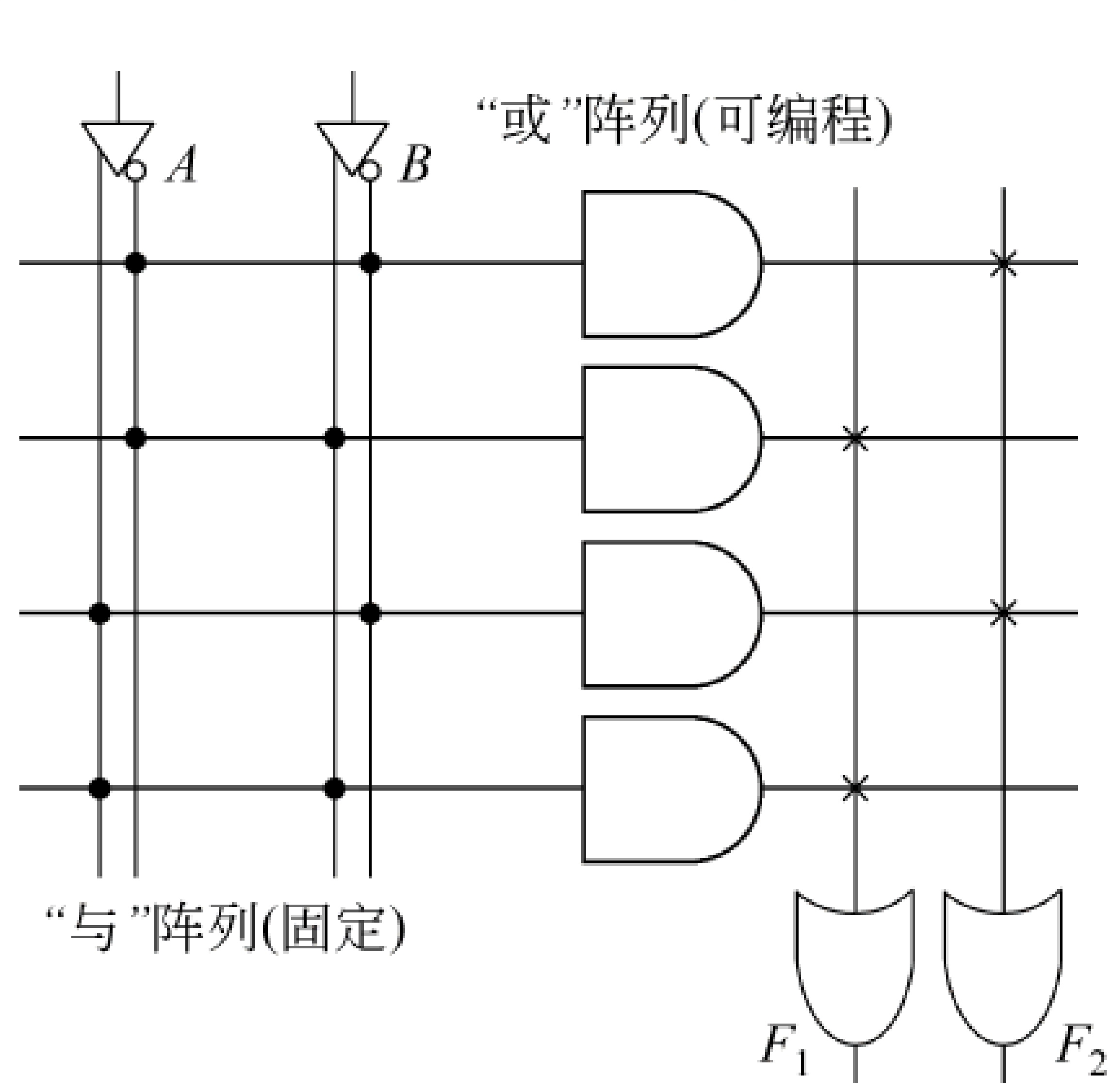


图 1-5　PROM 结构示意图

2. 可编程逻辑阵列(PLA)

图 1-6 是可编程逻辑阵列(Programmable Logic Array,PLA)的结构示意图,PLA 是一种“与”阵列和“或”阵列都可编程的逻辑阵列,这种“与”、“或”阵列都可编程的逻辑阵列编程时灵活度大,使用时也比较随意。图中完成的逻辑是:

$$F_1 = A\bar{B} + AB \qquad F_2 = \bar{A}\bar{B} + \bar{A}B$$

3. 可编程阵列逻辑(PAL)

图 1-7 是可编程阵列逻辑(Programmable Array Logic,PAL)的结构示意图,PAL 是一种“与”阵列可编程、“或”阵列固定的逻辑阵列。PAL 和简单 PROM 一样,逻辑结构相对简单,但是应用的灵活性不高。图中完成的逻辑是:

$$F_1 = \bar{A}B + AB \qquad F_2 = \bar{A}\bar{B} + A\bar{B}$$

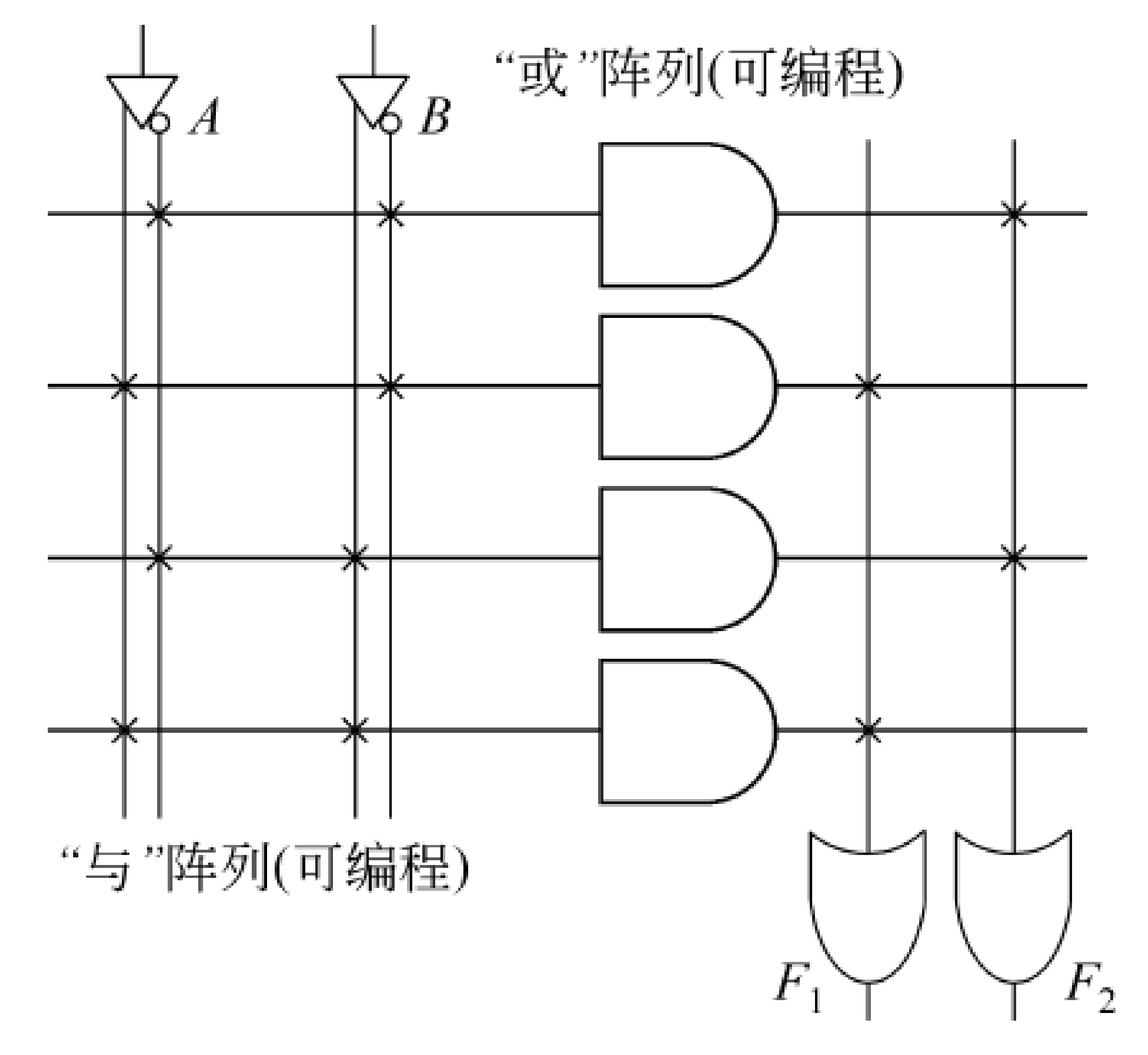


图 1-6　PLA 结构示意图

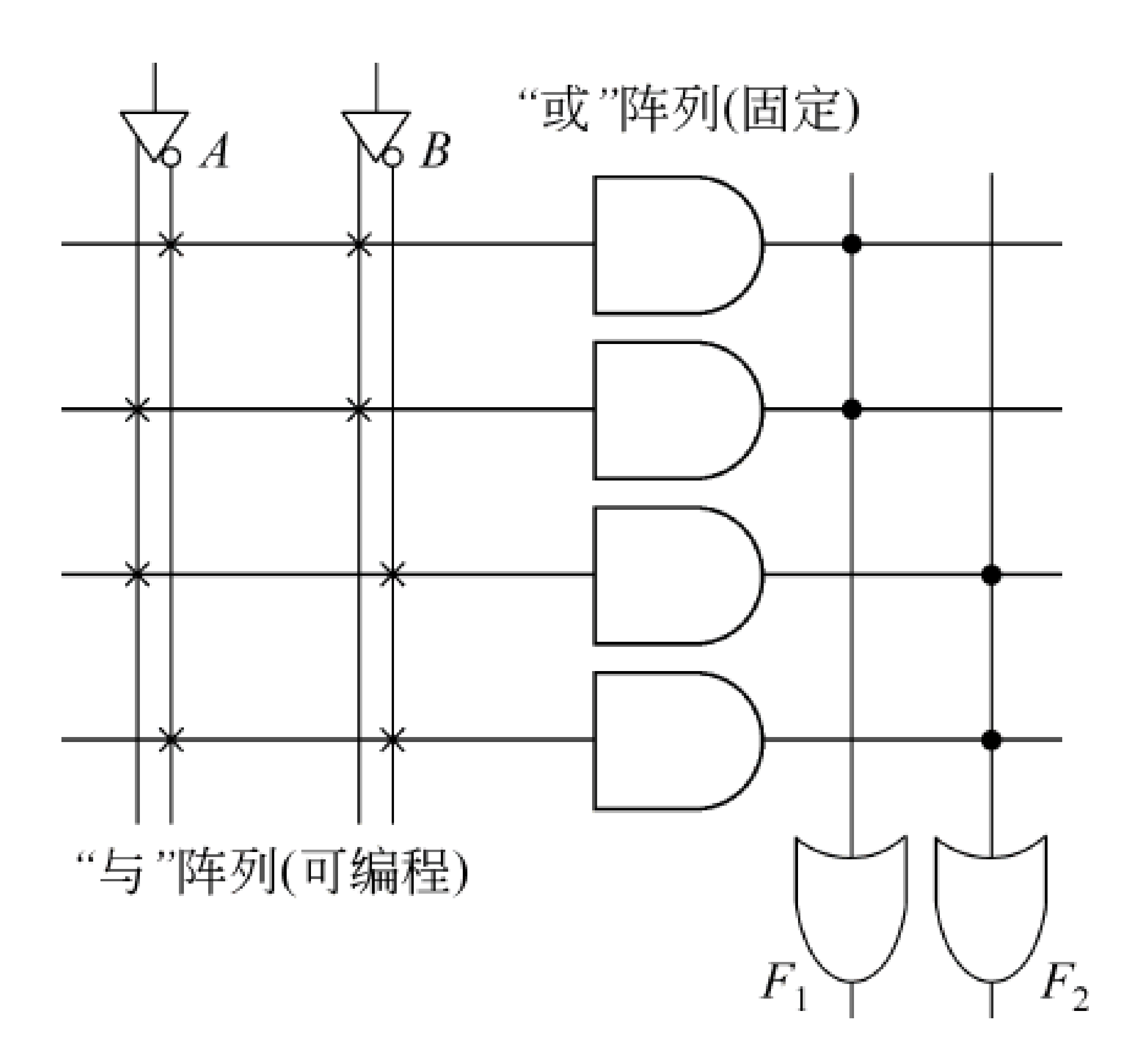


图 1-7　PAL 结构示意图

4. GAL16V8

1985 年 LATTICE 公司推出了一种新型的可编程逻辑器件——通用阵列逻辑(Generic Array Logic,GAL),GAL 器件与 PAL 的差别是 PAL 只能进行一次编程,而

GAL 可多次电擦写。此外，GAL 的输出端设置了可编程的输出逻辑宏单元(Output Logic Macro Cell，OLMC)，通过编程可将 OLMC 设置成不同的工作状态，有锁存、同步输出、异步输出、置'1'、清'0'等功能，从而增强器件的通用性。

图 1-8 是常见的 GAL16V8 的电路结构图。它有 1 个 32×64 位的可编程与逻辑阵列，8 个 OLMC，10 个输入缓冲器，8 个三态输出缓冲器和 8 个反馈/输入缓冲器。

"与"逻辑阵列的每个交叉点上设有编程电源，是可编程的；在 GAL16V8 中除了"与"逻辑阵列以外还有一些编程单元，用于完成电擦写、重复编程和设置加密位等功能。输出逻辑宏单元(OLMC)中包含或门、D 触发器、数据选择器以及一些门电路组成的控制电路，组成"或"逻辑阵列的 8 个或门就包含于 OLMC 中，它们和"与"逻辑阵列的连接是固定的；通过对 OLMC 内的结构控制字编程，便可设定 OLMC 的工作模式，得到不同类型的输出电路结构。

1.1.3 CPLD 器件

简单可编程器件通常用于实现规模较小的数字电路，芯片的输入和输出引脚数以及乘积项的个数都很有限，如果应用于大型的电路中就需要使用多个简单 PLD，为了解决这一问题，复杂可编程逻辑器件(Complex Programmable Logic Device，CPLD)应运而生。

图 1-9 为 ALTERA 公司的 MAX 7000 系列的 CPLD 结构框图。MAX 7000 系列器件由三个主要的部分组成：逻辑阵列块(Logic Array Block，LAB)、I/O 控制块和可编程互联阵列(Programmable Interconnect Array，PIA)。每个器件内含有若干个逻辑阵列块 LAB，每个 LAB 由 16 个宏单元(Macrocell)构成，宏单元是 CPLD 的基本结构，它相当于一个类似 PAL 的电路模块，用以实现基本的逻辑功能。I/O 控制块和芯片的输入输出引脚相连，可编程互联阵列 PIA 连接所有的宏单元和输入输出引脚，进行信号传递。图中的 INPUT/GCLK1、INPUT/OE1、INPUT/OE2/GCLK2 和 INPUT/GCLRn 信号是全局时钟、清零和使能信号，它们通过 PIA 及专用连线和每个宏单元相连。

MAX 7000 系列 CPLD 的宏单元结构如图 1-10 所示。

每个宏单元主要由三个部分组成：逻辑阵列、乘积项选择矩阵和可编程寄存器。逻辑阵列是可编程的，可编程为"与"逻辑；乘积项选择矩阵是一个"或"阵列，两者一起完成组合逻辑电路。图 1-10 中的右侧是可编程寄存器，可以编程实现 D 触发器、T 触发器、JK 触发器或钟控 SR 触发器。当电路中不需要触发器时，这一部分也可以被旁路掉，只完成组合逻辑功能。

1.1.4 FPGA 器件

现场可编程门阵列(Field-Programmable Gate Array，FPGA)是另一种集成度更高的复杂可编程逻辑器件。FPGA 的内部结构和 CPLD 的内部结构迥然不同，FPGA 内部没有"与"和"或"阵列。FPGA 通常包含三类可编程资源：逻辑单元(Logic Element，LE)、I/O 块和内部互连。逻辑单元是实现用户功能的基本单元，排列成二维阵列，分布于整个芯片；I/O 充当芯片上的逻辑与外部封装引脚的接口，围绕着逻辑单元阵列排列于芯片四周；内部互连包括各种长度的连接线段和一些可编程连接开关，它们将各个逻辑单元或 I/O 块连接起来，构成特定功能的电路。

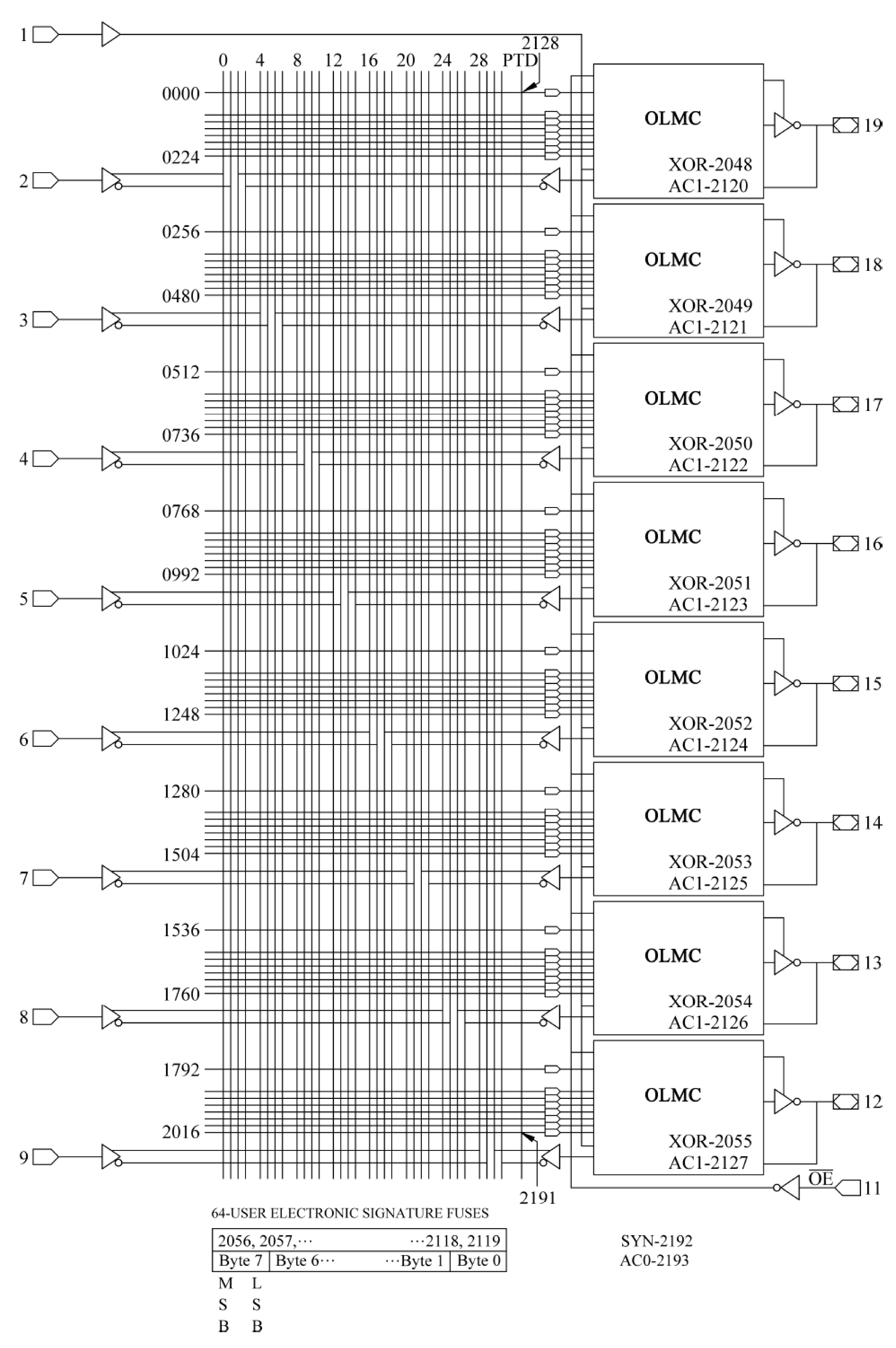


图 1-8　GAL6V8 的电路结构图

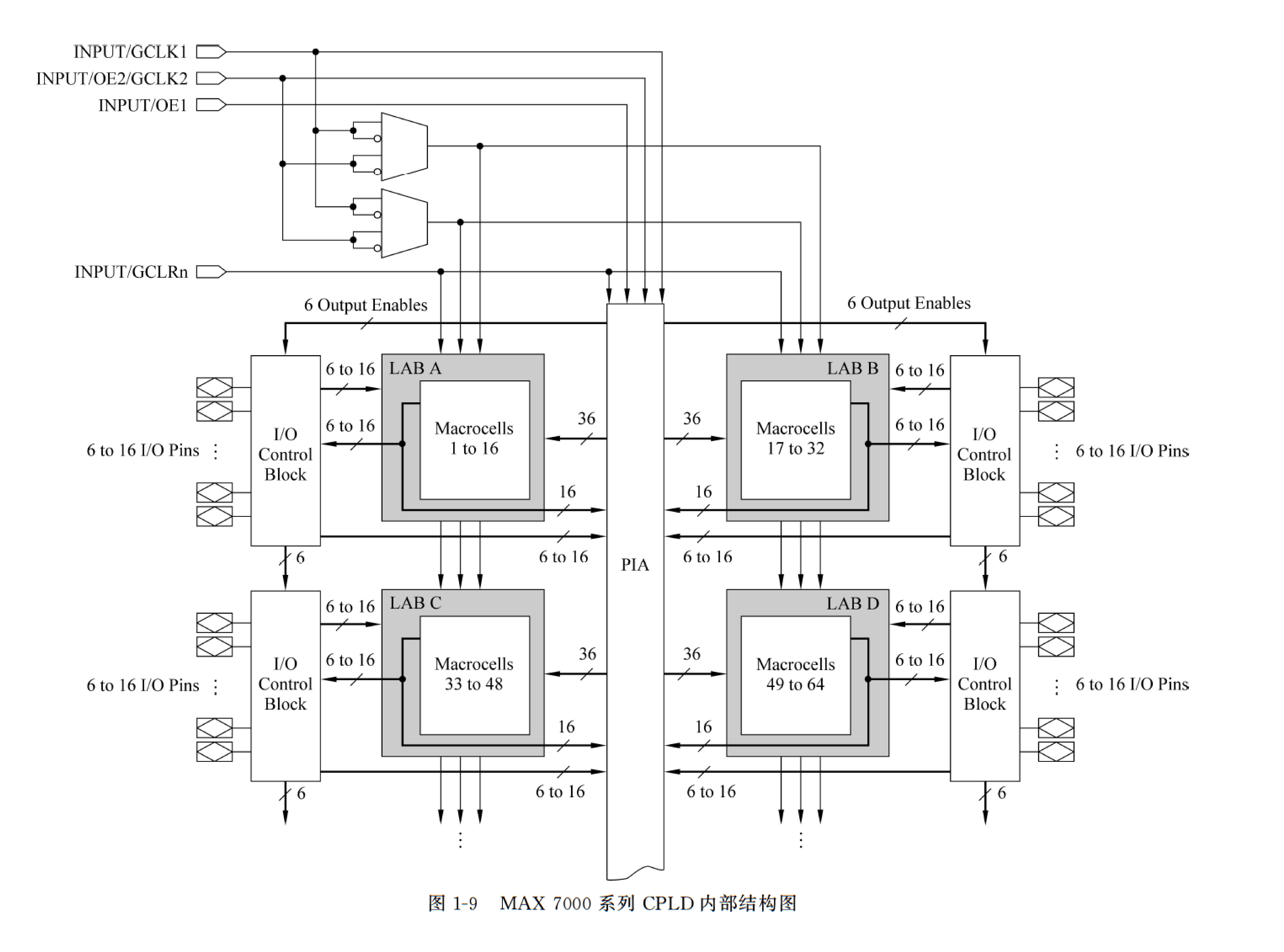


图 1-9　MAX 7000 系列 CPLD 内部结构图

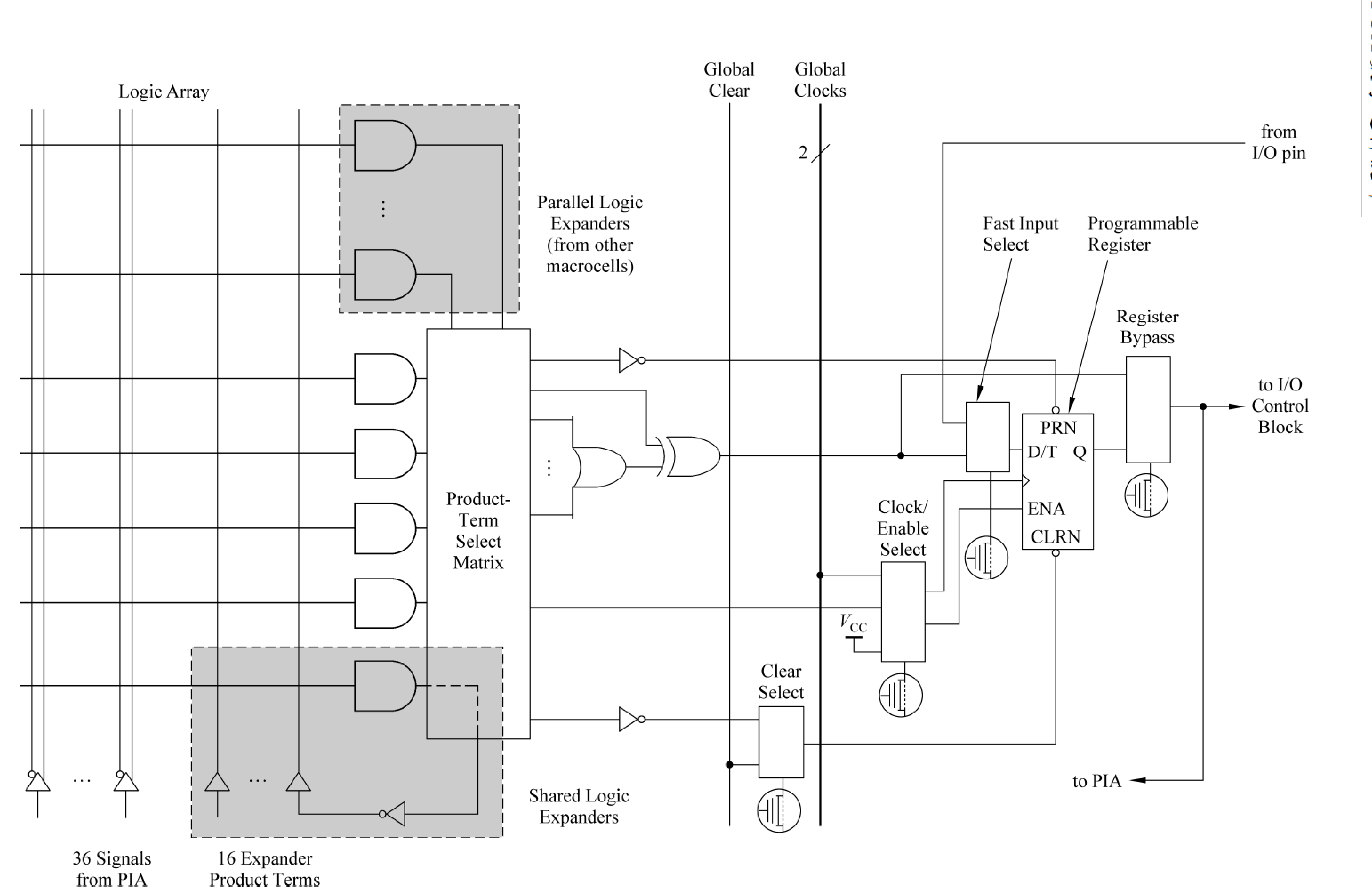


图 1-10 MAX 7000E 和 MAX 7000S 芯片宏单元图

图 1-11 是 Altera 公司 Cyclone Ⅱ系列 FPGA 内一个逻辑单元的普通工作模式原理图。一个 LE 主要由一个查找表(Look-Up Table,LUT)、一个寄存器以及进位和互联逻辑组成。LUT 本质是一个 RAM,包含若干个存储单元,用于实现一个小规模的逻辑函数,每个存储单元都可以存储一个逻辑 0 或 1,作为该单元的输出。目前 FPGA 中多使用 4 输入的 LUT,所以每一个 LUT 可以看成一个有 4 位地址线的 16×1 的 RAM,当用户通过原理图或者硬件描述语言(HDL)描述了一个逻辑电路以后,PLD/FPGA 开发软件会自动计算逻辑电路所有可能的结果,并且把结果事先写入每个存储单元(RAM)中。这样,对每输入一个信号进行逻辑运算,就相当于输入一个地址进行查表,找出地址对应的内容,即为此组逻辑信号的运算结果,然后输出即可。

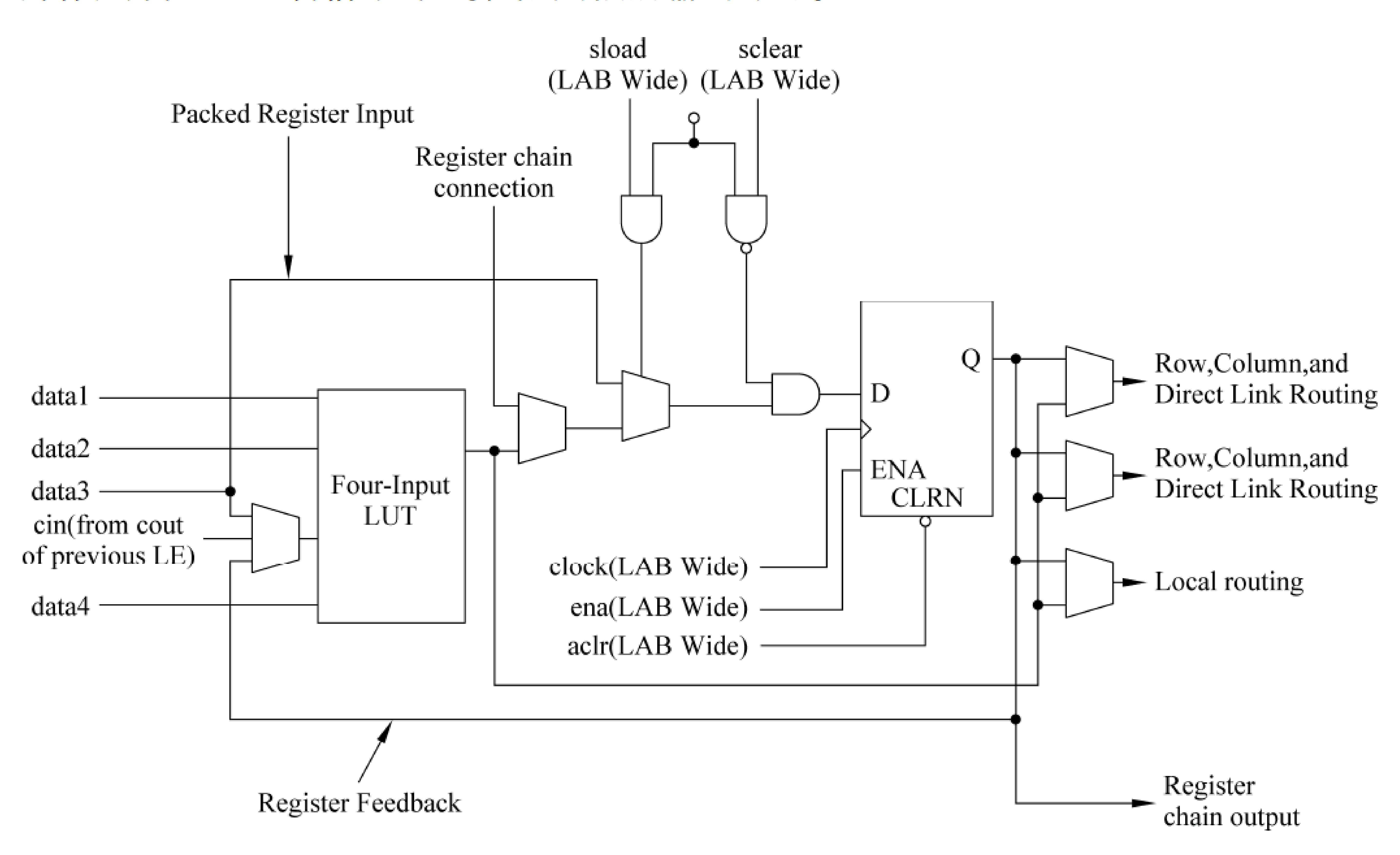


图 1-11 Cyclone Ⅱ系列 FPGA 内逻辑单元的普通工作模式原理图

1.1.5 专用集成电路

FPGA 的可编程特性使其应用非常灵活,但正是因为其可编程,反而降低了芯片内部逻辑门的使用率,导致 FPGA 的功耗高、速度慢、资源冗余和价格昂贵。专用集成电路芯片(Application-Specific Integrated Circuit,ASIC)是一种为专门目的而设计的集成电路,是指应特定用户要求和特定电子系统的需要而设计、制造的集成电路。ASIC 分为全定制和半定制。全定制设计需要设计者完成所有电路的设计,如果设计较为理想,全定制的 ASIC 芯片能够比半定制的 ASIC 芯片运行速度更快。半定制芯片使用逻辑单元库里的标准逻辑单元(Standard Cell),设计时可以从标准逻辑单元库中选择小规模集成电路(Small Scale Integrated Circuit,SSIC)、中规模集成电路(Medium Scale Integrated Circuit,MSIC)、数据通路、存储器、甚至系统级模块(如乘法器、微控制器等)和 IP 核等。门电路、触发器属于小规模集成电路,加法器、比较器属于中规模集成电路。这些逻辑单元已经布局完毕,而且设计得较为可靠,设计者可以比较方便地完成系统设计。ASIC 的

特点是面向特定用户的需求，ASIC 在批量生产时与通用集成电路相比，具有体积更小、功耗更低、可靠性更高、性能更高、保密性更强、成本降低等优点。

FPGA 和 ASIC 目前都是电子设计领域的主流产品，二者不同的技术特性决定了其不同的市场：ASIC 一般被用于批量大的专用产品中，FPGA 一般在小批量的产品设计中和产品开发阶段占据优势，用户在设计的时候可以灵活选择。

1.2 Cyclone Ⅱ系列 FPGA

1.2.1 概述

1.2.1.1 Cyclone Ⅱ系列 FPGA 简介

Altera 低成本 Cyclone Ⅱ系列 FPGA 是基于 1.2V、90nm 的 SRAM 制造工艺而设计的，每片芯片含高达 68K 个逻辑单元和 1.1Mb 片内 RAM(M4K RAM blocks)，片内还嵌入了支持高性能 DSP 应用的 18×18 位乘法器，用于管理系统时钟的锁相环(Phase-Locked Loops，PLL)和用于支持 SRAM 芯片及 DRAM 芯片的高速外部存储器接口。Cyclone Ⅱ器件还支持低成本应用中常见的各种外部存储器接口和 I/O 协议。Cyclone Ⅱ系列芯片以其优异的性价比在市场上得到了广泛的应用。

Cyclone Ⅱ系列芯片的主要性能见表 1-1。

表 1-1 Cyclone Ⅱ系列芯片性能

性　　能	EP2C5	EP2C8	EP2C20	EP2C35	EP2C50	EP2C70
LE 数目	4608	8256	18 752	33 216	50 528	68 416
M4K RAM blocks (4 Kbits plus 512 parity bits)	26	36	52	105	129	250
Total RAM bits	119 808	165 888	239 616	483 840	594 432	1 152 000
Embedded multipliers	13	18	26	35	86	150
PLLs	2	2	4	4	4	4
Maximum user I/O pins	158	182	315	475	450	622

1.2.1.2 Cyclone Ⅱ系列 FPGA 芯片结构

Cyclone Ⅱ系列芯片内部逻辑单元以行、列的形式排列成二维逻辑阵列，行、列之间含有大量的内部连线，它们在各个逻辑阵列块(Logic Array Blocks，LAB)、片内存储器和片内乘法器之间进行信号传递。

图 1-12 为 Cyclone Ⅱ EP2C20 的芯片内部框图，其他同系列的芯片的结构均与之类似。

逻辑阵列由 LAB 构成，每个 LAB 含有 16 个逻辑单元(LE)。LE 是一个用于具体完成逻辑功能的最小逻辑结构。LAB 排成的行列阵列分布于整个 FPGA 芯片，Cyclone Ⅱ系列芯片中含有 4608～68 416 个 LE。

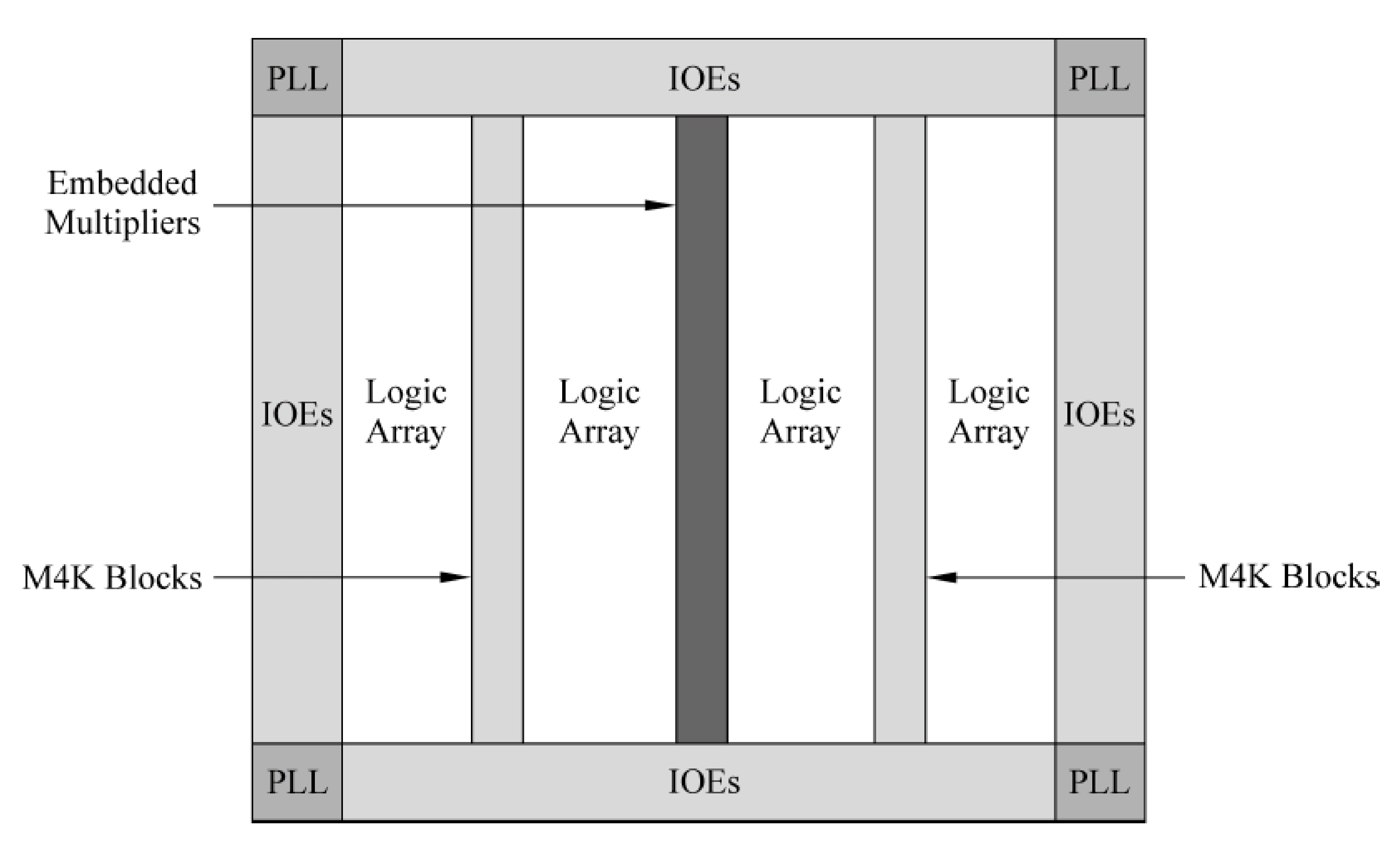


图 1-12 Cyclone Ⅱ EP2C20 芯片结构框图

Cyclone Ⅱ系列芯片提供了一个全局时钟网和 4 个锁相环 PLL，全局时钟网由 16 个全局时钟线构成，用于驱动整个芯片的系统时钟。全局时钟网可以为芯片内的所有单元提供时钟驱动，例如输入输出单元(Input/Output Elements，IOE)、LE、片内乘法器和片内存储器块等。Cyclone Ⅱ系列芯片内的 PLL 用于提供通用时钟，例如时钟同步、相移以及支持高速差分 I/O 的外部输出等。

M4K 存储器块是真正的双端口存储器，含有 4Kb 存储空间和奇偶校验位(总共 4608 位)，这些存储器块可作为真正双端口模式、简单双端口模式和存取速度可达 250MHz 的 36 位宽单端口模式使用。这些模块在芯片中按列分布，且都分布在芯片中两列 LAB 的中间，Cyclone Ⅱ系列芯片总共含有 119～1152Kb 的片内存储器。

每个片内乘法器都可以作为一个 18×18 位的乘法器和两个 9×9 位的乘法器使用，其工作频率可高达 250MHz。片内乘法器在器件内也按列排放在两列 LAB 的中间。

输入/输出单元排列在 Cyclone Ⅱ 芯片各行/列的末端，分布在芯片的四周，和 I/O 引脚相连，支持各种常见的单端或差分 I/O 协议。

表 1-2 展示了 Cyclone Ⅱ系列器件的资源，具体包括各 Cyclone Ⅱ系列芯片内部的片内 M4K 存储器、片内乘法器、PLL 和芯片内部逻辑部件排成的行、列数目。

表 1-2 Cyclone Ⅱ器件资源

芯片型号	LAB 列	LAB 行	LE	PLL	M4K RAM	片内乘法器
EP2C5	24	13	4608	2	26	13
EP2C8	30	18	8256	2	36	18
EP2C20	46	26	18 752	4	52	26
EP2C35	60	35	33 216	4	105	35
EP2C50	74	43	50 528	4	129	86
EP2C70	86	50	68 416	4	250	150

1.2.2 逻辑单元

逻辑单元(LE)是 Cyclone Ⅱ芯片中构成用户逻辑电路的最小逻辑结构,它结构紧凑,使用灵活,可用来构成各种复杂逻辑电路。逻辑单元有以下主要特性:

(1) 一个4输入的查找表(Look-Up Table,LUT),相当于一个函数发生器,可以用来完成任意一个4变量的组合逻辑电路。

(2) 一个可编程寄存器。

(3) 一个进位链连接。

(4) 一个寄存器链连接。

(5) 能够驱动所有类型的互联:本地互联、行互联、列互联、寄存器链互联和直接互联。

(6) 支持寄存器打包(register packing)模式。

(7) 支持寄存器反馈。

图1-13显示的是一个 Cyclone Ⅱ的逻辑单元。

每一个 LE 的可编程寄存器都可以配置为 D 触发器、T 触发器、JK 触发器或 RS 触发器,每一个寄存器都有 data、clock、clock enable 和 Clear 等输入端。来自全局时钟网、通用 I/O 引脚或者任何内部逻辑的信号都可以驱动寄存器的 clock 和 Clear 控制信号。通用 I/O 引脚和内部逻辑信号都能够驱动 clock enable 端。作为组合逻辑使用时,寄存器被旁路,由查找表直接驱动 LE 输出。

每个 LE 有三个输出端,分别用于驱动本地、行和列布线资源(routing resources)。LE 的三个输出端,其中两个用来驱动行、列及直接布线,一个用来驱动本地互联布线。不论是 LUT 还是寄存器都可以单独驱动这三个输出端,例如 LUT 驱动一个输出端而寄存器驱动另外两个,这一特性称为寄存器打包。可以让查找表和寄存器分别完成相互独立的功能,这样提高了芯片的应用性能。当作为寄存器打包模式使用时,LAB 同步加载信号无效。

1.2.3 片内存储器

Cyclone Ⅱ 片内存储器由多列 M4K 存储器构成,M4K 存储器块内含有同步写入的输入寄存器和输出寄存器,用于流水线设计和提高系统性能。输出寄存器可以旁路,但是输入寄存器不能。每个 M4K 存储器块可以有不同的配置方法,包括真双口 RAM、简单双口 RAM、单口 RAM、ROM 或者 FIFO 缓存。M4K 存储器具有如下特性:

(1) 4068位 RAM。

(2) 250MHz 工作频率。

(3) 真正双口存储器。

(4) 简单双口存储器。

(5) 单口存储器。

(6) 字节使能。

(7) 奇偶校验位。

(8) 移位寄存器。

(9) FIFO 缓存。

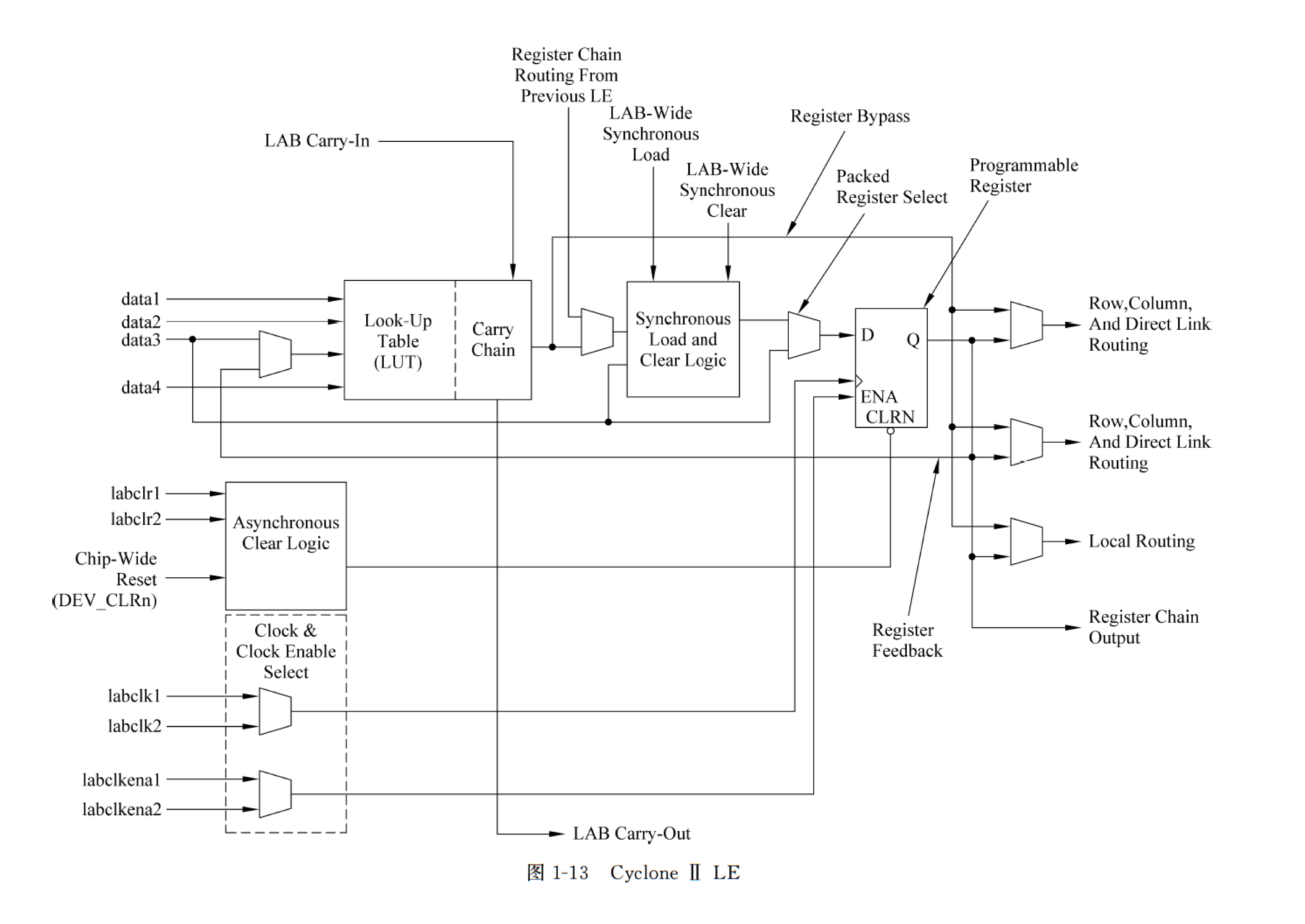


图 1-13　Cyclone Ⅱ LE

(10) ROM。

(11) 多种时钟模式。

(12) 地址时钟使能。

表 1-3 显示的是 Cyclone Ⅱ 系列芯片内的 M4K 存储器数量和布局情况。

表 1-3　Cyclone Ⅱ 系列芯片内的 M4K 存储器数量和布局情况

芯　　片	M4K 列数	M4K 个数	总计 RAM 位数
EP2C5	2	26	119 808
EP2C8	2	36	165 888
EP2C20	2	52	239 616
EP2C35	3	105	483 840
EP2C50	3	129	594 432
EP2C70	5	250	1 152 000

表 1-4 总结了 M4K 存储器的特性。

表 1-4　M4K 存储器的特性

特　　性	说　　明
最高性能	250MHz
每个 M4K 的总 RAM 位(含校验位)	4608
支持的配置	4K×1 2K×2 1K×4 512×8 512×9 256×16 256×18 128×32(真双口下无效) 128×36(真双口下无效)
奇偶校验位	每个字节带一个奇偶校验位,保证数据传输的正确性
字节使能	M4K 存储器支持 1、2、4、8、9、16、18、32 和 36 位的数据写入。字节使能端允许输入数据屏蔽为特定的数据宽度,只改变写入字节的内容,其他未写入字节保持不变
打包模式	如果两个单口存储器都为单时钟模式,且大小都等于或者小于 M4K RAM 的一半,那么这两个 RAM 块可以组合为一个 M4K 使用
地址时钟使能	M4K 存储器支持地址时钟使能,在时钟使能信号有效之前,可以一直保持之前的地址值。这一特征用于处理 cache 应用中的数据缺失问题
寄存器初始化文件(.mif)	当 M4K 存储器被设置为 RAM 或 ROM 使用时,可以用这个文件初始化存储器中的内容
加电条件	加电时,输出块被清零
寄存器清零	只对输出寄存器清零
写时同端口读	新数据在时钟上升沿有效
写时混合端口读	旧数据在时钟上升沿有效

当清零信号作用于输入寄存器时，片内存储器的异步清零信号立即对输入寄存器进行清零。然而，输出存储器块直到下一个时钟边沿到来时才清零。而当清零信号作用于输出寄存器时，片内存储器的异步清零信号对输出寄存器立即进行清零。

表1-5总结了M4K存储器支持的不同存储器工作模式。

表1-5　M4K存储器支持的不同存储器工作模式

存储器模式	说　　明
单口存储器	当不需要同时读写时，M4K存储器可置为单口模式
简单双口存储器模式	简单双口模式支持同时读写(一读一写)
混合宽度的简单双口存储器模式	读写使用不同的数据宽度的简单双口模式
真双口存储器模式	真双口模式支持任何组合的双口操作：两个读口、两个写口和两个不同时钟频率下的一个读口和一个写口
混合宽度的真双口存储器模式	读写使用不同的数据宽度的真双口模式
片内移位寄存器	此模式下，时钟下降沿写数据，上升沿读数据
ROM	工作于ROM模式，ROM中的内容由.mif文件初始化
FIFO缓冲器	在M4K存储器中可以实现单时钟或双时钟的FIFO，对于空的FIFO不允许进行同时读写

表1-6总结了M4K存储器支持的不同时钟工作模式。

表1-6　M4K存储器支持的不同时钟工作模式

时钟模式	说　　明
独立时钟模式	这种模式下，M4K存储器的每个端口(端口A和端口B)都有一个独立的时钟单独控制，时钟A控制端口A的所有寄存器，时钟B控制端口B的所有寄存器
输入输出时钟模式	两个时钟A和B，一个对所有的输入寄存器进行控制，例如数据输入、写使能和地址；另一个控制M4K存储器的所有输出寄存器
读写时钟模式	此模式下有两个时钟可用，写时钟控制数据输入、写地址和写使能；读时钟控制数据输出、读地址和读使能
单时钟模式	这种模式下，一个单独的设置和时钟使能信号一起用于控制存储器块的所有寄存器，此时不支持寄存器异步清零

表1-7显示了M4K存储器块在不同工作模式下支持的时钟工作模式。

表1-7　M4K存储器块在不同的存储器工作模式下支持的时钟工作模式

时钟模式	真双口模式	简单双口模式	单口模式
独立时钟模式	√		
输入输出时钟模式	√	√	√
读写时钟模式		√	
单时钟模式	√	√	√

1.2.4 片内乘法器

CycloneⅡ系列器件在芯片内部嵌入了乘法器，提高了FPGA的数字信号处理能力，可进行有限脉冲响应(FIR)滤波、快速傅里叶变换(FFT)和离散余弦变换(DCT)等运算。片内乘法器有两种使用形式：

(1) 一个18位的乘法器。

(2) 两个独立的9位乘法器。

片内乘法器的工作频率可高达250MHz。每一个CycloneⅡ芯片都有1～3列的片内乘法器。每个乘法器和一个LAB等高。表1-8列出了CycloneⅡ芯片的乘法器个数。

表1-8 CycloneⅡ芯片的乘法器个数

芯　片	乘法器列数	乘法器个数	9×9 乘法器	18×18 乘法器
EP2C5	1	13	26	13
EP2C8	1	18	36	18
EP2C20	1	26	52	26
EP2C35	1	35	70	35
EP2C50	2	86	172	86
EP2C70	3	150	300	150

片内乘法器含有以下结构(如图1-14所示)：

(1) 乘法时钟。

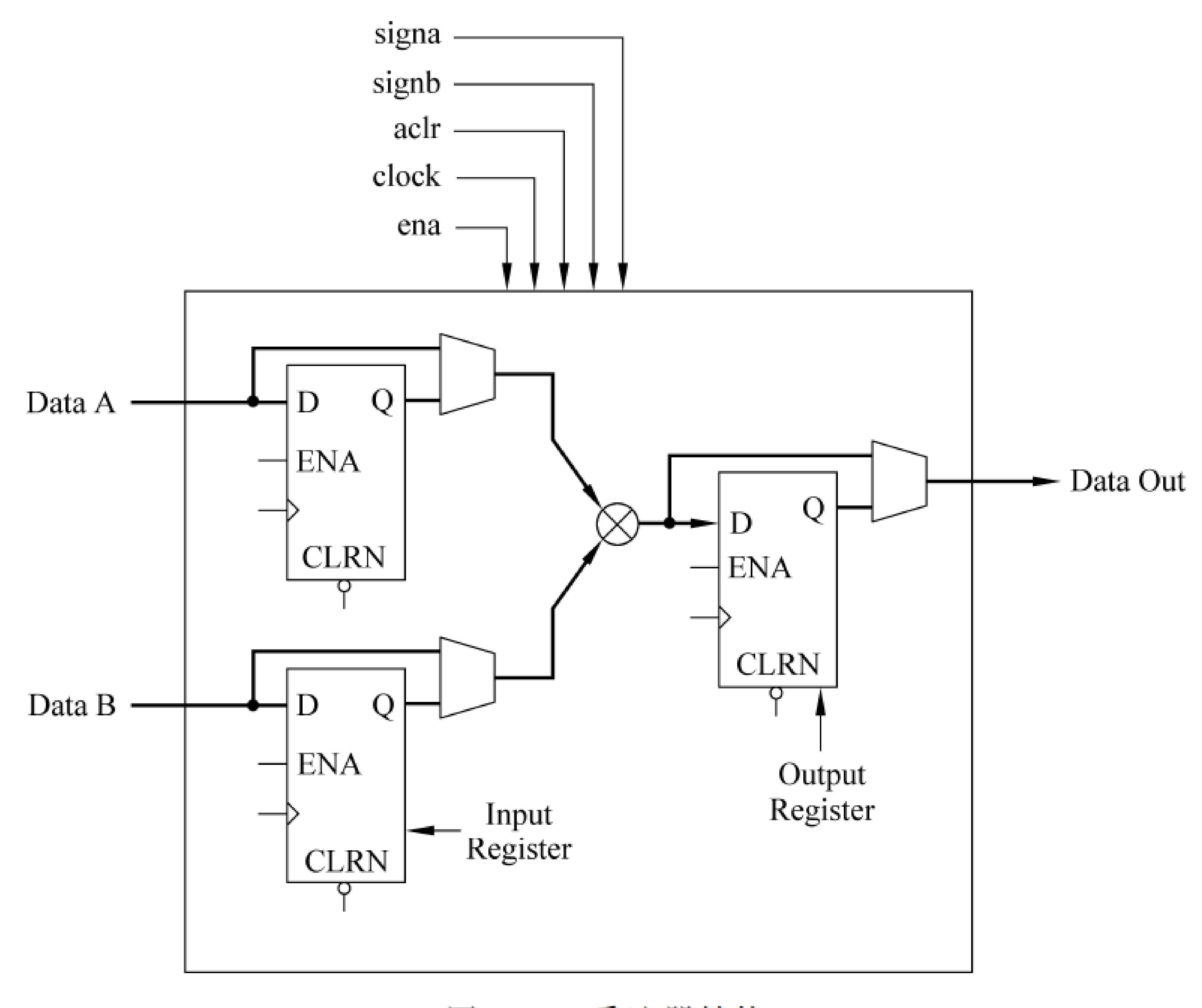


图1-14 乘法器结构

（2）输入和输出寄存器。

（3）输入和输出接口。

片内乘法器的两个数据输入端 Data A 和 Data B 相互独立，可以分别输入有符号数和无符号数，它们分别由 signa 和 signb 控制，'1'表示是有符号数，'0'表示是无符号数。当两个输入端同为无符号数时，输出为无符号数；否则，如果有一个输入端是有符号数，那么输出也是有符号数。每个乘法器的输入端只有一个 signa 和 signb 控制，因此在 9×9 的模式下，Data A 和 Data B 的两个输入端（两个 9×9）的符号设定必须一样（同为有符号数或者无符号数）。

每一个片内乘法器有 5 个控制信号，其中 signa 和 signb 用于设置输入数据的符号属性，另外的 clk、ena 和 aclr 用于控制一个乘法器内的所有寄存器。

1.2.5 输入输出模块

输入输出模块（IOE）支持如下特性：

（1）差分和单端 I/O 标准。

（2）3.3V、64 和 32 位、66 和 33MHz 的 PCI 接口。

（3）支持 JTAG（Joint Test Action Group）和 BST（Boundary-Scan Test）协议。

（4）很强的输出驱动控制能力。

（5）配置时只需要很小的上拉电阻。

（6）三态缓冲。

（7）总线状态维持电路（Bus-hold circuitry）。

（8）应用模式时，上拉电阻可编程。

（9）输入输出延时可编程。

（10）漏极开路输出。

（11）DQ 和 DQS I/O 引脚。

（12）VREF 引脚。

Cyclone Ⅱ系列 FPGA 的 IOE 含有一个双向 I/O 缓冲器和三个寄存器，用于完成内部单一双向的数据传送。图 1-15 显示了 Cyclone Ⅱ的 IOE 结构。IOE 包含一个输入寄存器，一个输出寄存器和一个输出使能寄存器。输入寄存器可以用来加快 setup 时间，输出寄存器可以用来加快 clock-to-output 时间，另外，输出使能寄存器也可以用来加快 clock-to-output enable 时间。IOE 可以设置为输入、输出或者双向引脚。

Cyclone Ⅱ器件支持多种单端和差分 I/O 标准，以及大量的接口和协议，给系统开发人员提供了系统设计的灵活性。

Cyclone Ⅱ器件支持单端 I/O 标准（例如 LVTTL、LVCMOS、SSTL-2、SSTL-18、HSTL-18、HSTL-15、PCI 和 PCI-X 标准）以连接板上的其他器件。当 FPGA 与其他高级存储器件，如双倍数据速率（DDR 和 DDR2）SDRAM 和 QDRⅡ SRAM 器件，一起工作时，单端 I/O 标准将成为关键因素。表 1-9 列出了 Cyclone Ⅱ器件内的单端 I/O 标准和所支持的目标性能。

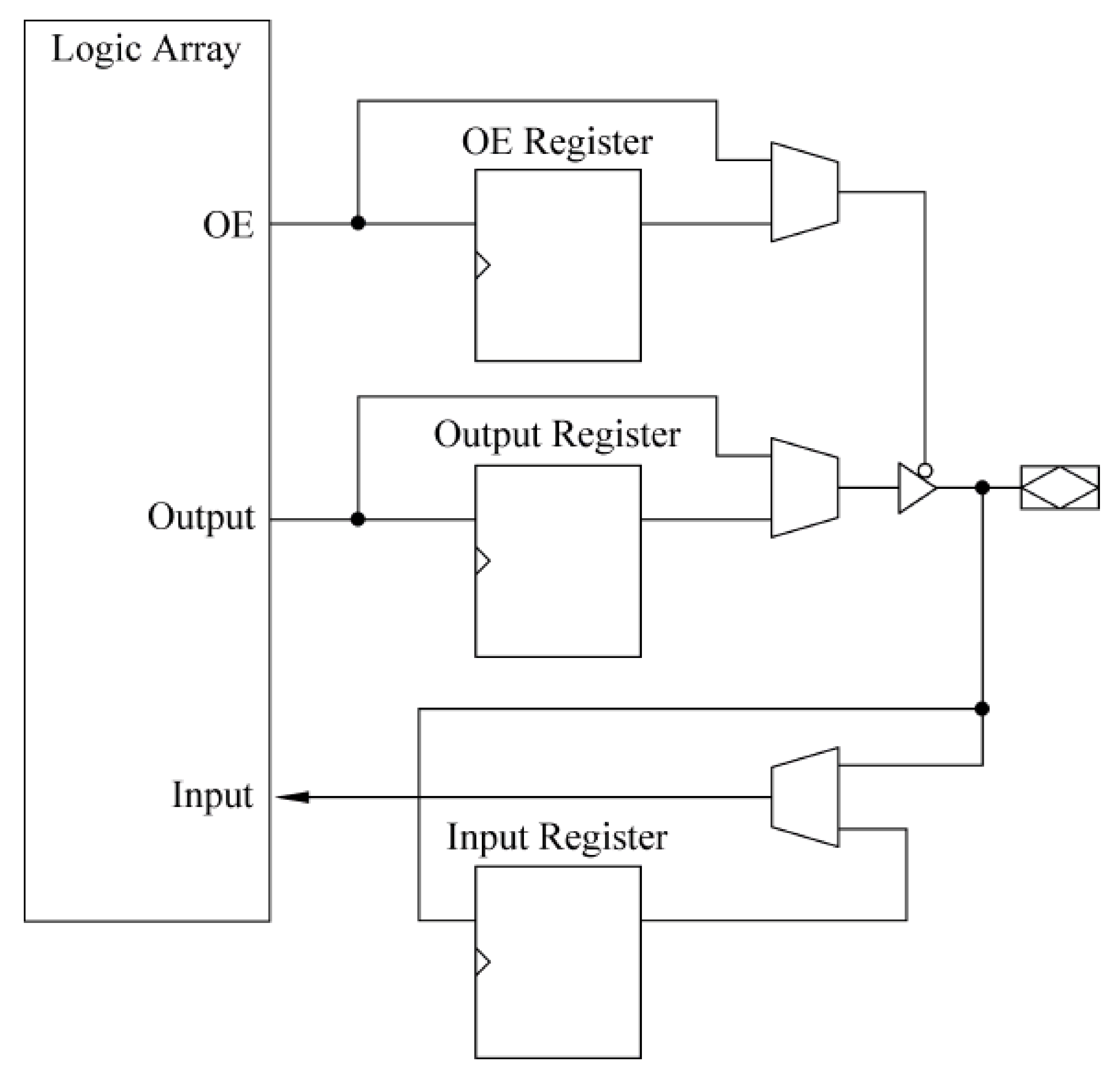


图 1-15　Cyclone Ⅱ 的 IOE 结构

表 1-9　Cyclone Ⅱ 器件单端 I/O 标准支持

I/O 标准	性　　能	典 型 应 用
3.3-V/2.5-V/1.8-V LVTTL	167MHz	通用
3.3-V/2.5-V/1.8-V/1.5-V LVCMOS	167MHz	通用
3.3-V PCI	66MHz	个人电脑(PC),嵌入式应用
3.3-V PCI-X	100MHz	PC,嵌入式应用
2.5-V/1.8-V SSTL Class Ⅰ	167MHz	存储器
2.5-V/1.8-V SSTL Class Ⅱ	133MHz/125MHz	存储器
1.8-V/1.5-V HSTL Class Ⅰ	167MHz	存储器
1.8-V/1.5-V HSTL Class Ⅱ	100MHz	存储器

与单端 I/O 标准相比,Cyclone Ⅱ 器件的差分信号提供更好的噪音容限,产生更低的电磁干扰(EMI),并降低了功耗。表 1-10 列出了 Cyclone Ⅱ 器件内的差分 I/O 标准和所支持的目标性能。

表 1-10　Cyclone Ⅱ 器件的差分 I/O 标准支持

I/O 标准	性　　能	典 型 应 用
LVDS	805Mbps 接收端, 622Mbps 发送端	芯片到芯片接口应用,背板驱动
mini-LVDS	170Mbps	通用
RSDS	170Mbps	通用
LVPECL	150MHz	只用于时钟输入
差分 HSTL	167MHz	存储器
差分 SSTL	167MHz	存储器

在新的以及现有的 FPGA 市场上,Cyclone Ⅱ 器件扩展了 FPGA 在低成本、大批量应

用领域的角色。FPGA 如今不再仅限于外围应用,可以在系统中执行很多关键性处理任务。随着 FPGA 越来越多地应用于系统的数据路径,当系统存储需求超过片内丰富的存储器资源时,FPGA 必须具有和外部存储器件的接口。

Altera 基于成功的 Cyclone 器件系列,通过和业界领先的存储器供应商合作,确保了用户能够将最新的存储器器件连接至 Cyclone Ⅱ系列 FPGA。Cyclone Ⅱ器件被设计成为能够通过一个专用的接口,和双倍数据速率(DDR)、DDR2、单倍速率(SDR)SDRAM 器件以及四倍数据速率(QDRII)SRAM 器件进行通信,保证快速可靠的数据传输,传输速率最高达到 668Mbps。开发人员可以在几分钟里集成 SDRAM 和 SRAM 器件到他们的系统中,和基于 Cyclone Ⅱ优化的、现成的 IP 控制器核一起运行。表 1-11 总结了 Cyclone Ⅱ外部存储器接口支持。

表 1-11　Cyclone Ⅱ器件支持的外部存储器接口

存储技术	I/O 标准	最大总线宽	最大时钟速度	最大数据速率
SDR SDRAM	3.3-V LVTTL	72b	167MHz	167Mbps
DDR SDRAM	2.5-V SSTL Class Ⅰ,Ⅱ	72b	167MHz	334Mbps
DDR2 SDRAM	1.8-V SSTL Class Ⅰ,Ⅱ	72b	167MHz	334Mbps
QDRII SRAM	1.8-V HSTL Class Ⅰ,Ⅱ	36b	167MHz	668Mbps

1.3　DE-70 开发平台

1.3.1　外观和组件

图 1-16 所示为 DE2-70 开发板的照片,图中标出了开发板的布局以及板上关键组件的位置。

DE2-70 开发板为用户提供了大量组件,使得开发者可以轻松地实现多种多媒体设计项目。

DE2-70 开发板提供了如下硬件配置:

(1) Altera Cyclone Ⅱ系列 2C70 FPGA 芯片。

(2) Altera 系列配置芯片 EPCS16。

(3) 用于编程和 API 控制的 USB Blaster(在板上);支持 JTAG 和 Active Serial(AS)两种编程模式。

(4) 2MB SSRAM。

(5) 2 个 32MB SDRAM。

(6) 8MB Flash memory。

(7) SD Card 槽。

(8) 4 个按钮开关。

(9) 18 个拨动开关。

(10) 18 个红色 LED。

(11) 9 个绿色 LED。

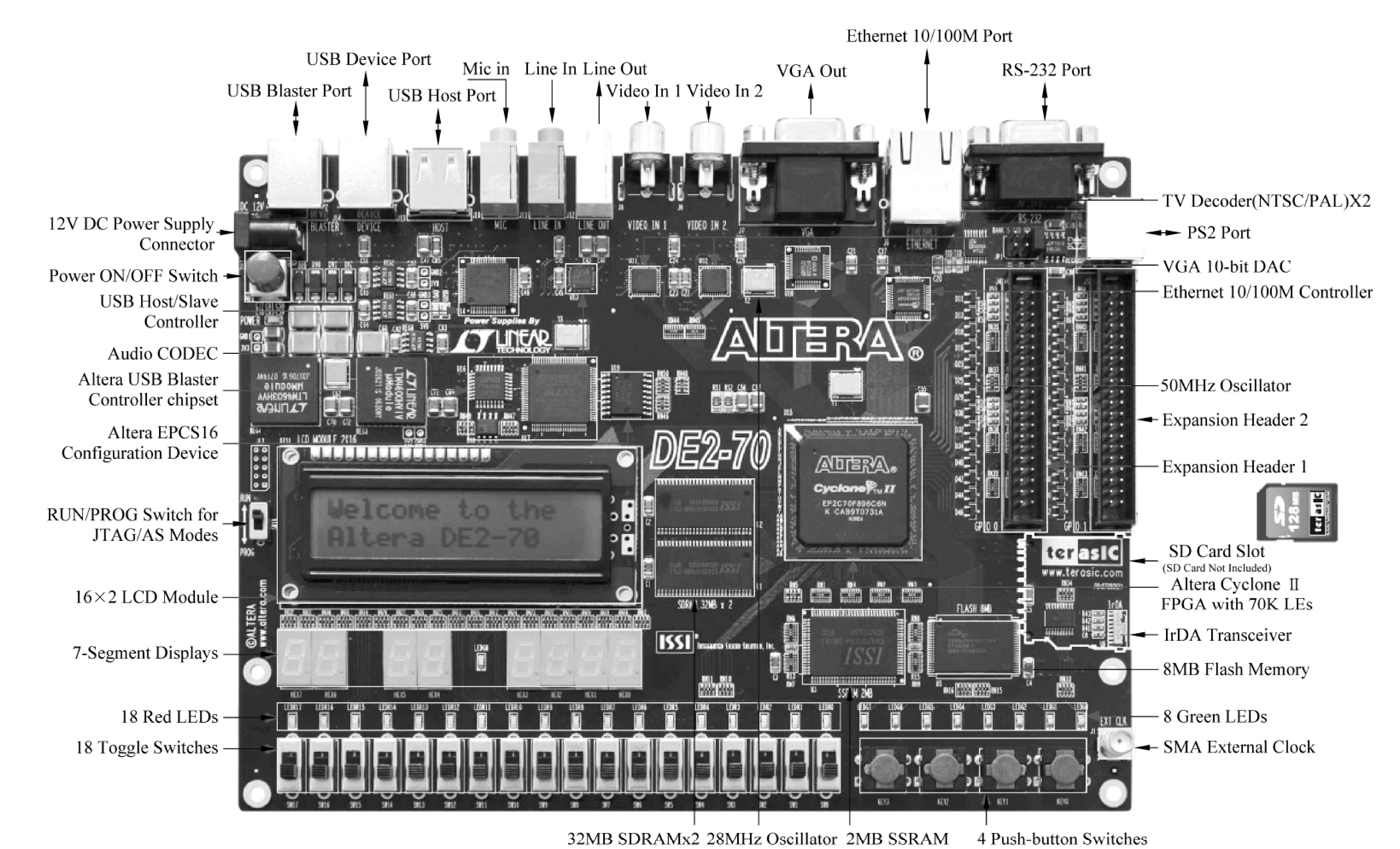


图 1-16　DE2-70 开发板

(12) 50MHz 振荡器和 28.63MHz 振荡器提供时钟源。

(13) 24b CD 音频 CODEC(编码/解码器),并带有 line-in、line-out 和 microphone-in 接口。

(14) VGA DAC(数模转换器),并带有 VGA-out 接口。

(15) 2 个 TV 解码器(NTSC/PAL/SECAM)和 TV-in 接口。

(16) 10/100 网络控制器并带有接口。

(17) USB Host/Slave 控制器,并带有 USB A 型和 B 型接口。

(18) RS-232 接口。

(19) PS/2 鼠标/键盘接口。

(20) IrDA 转换器。

(21) 1 个 SMA 接口。

(22) 2 个带有二极管保护的 40 个引脚外部扩展接口。

图 1-17 显示了 DE2-70 的结构框图,为了便于使用,所有的外围部件都通过 FPGA 连接起来,这样,用户就可以通过配置 FPGA 来完成所有系统的控制。

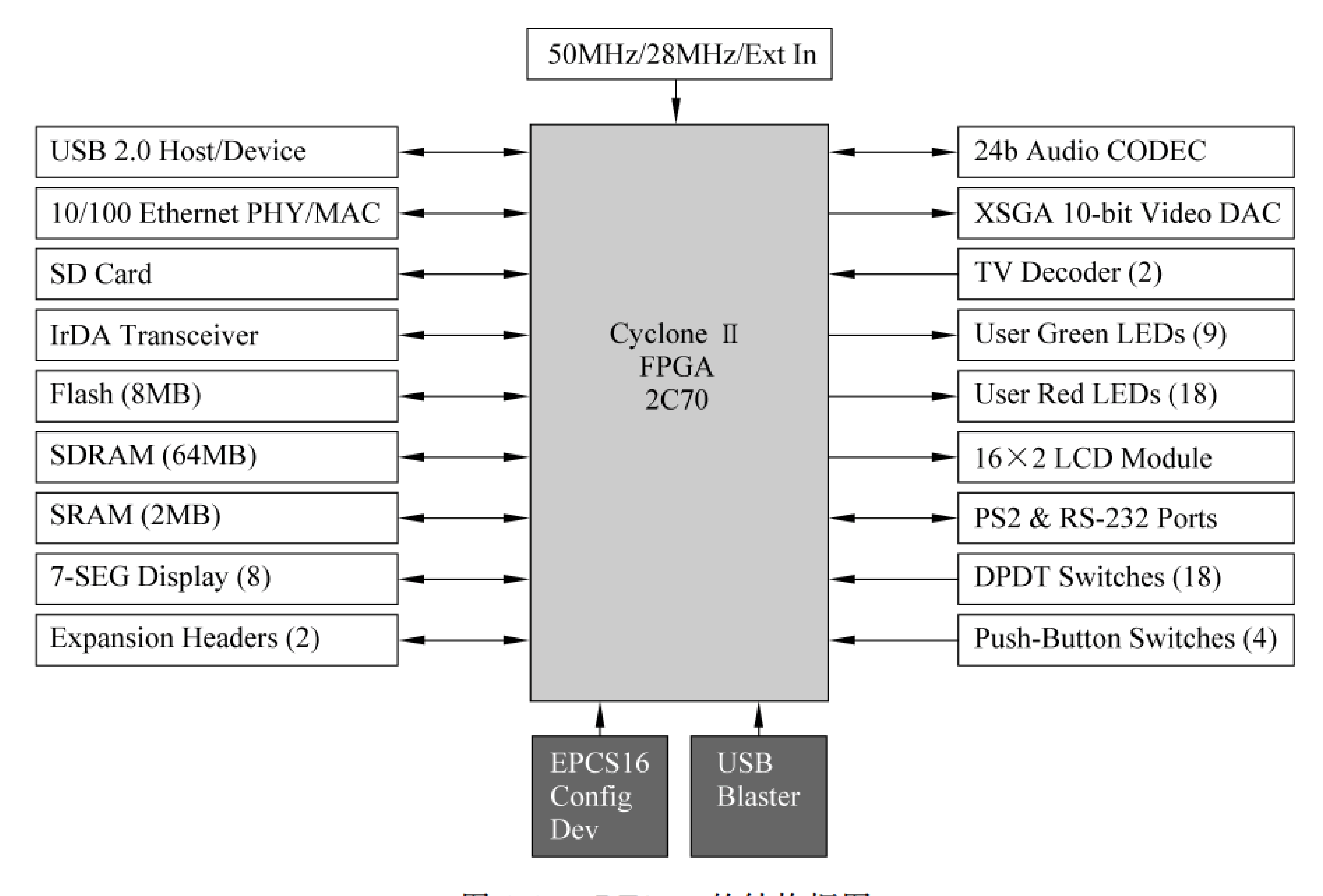


图 1-17 DE2-70 的结构框图

在图 1-17 中,各个部件的简单特性介绍如下:

(1) Cyclone Ⅱ 系列 2C70 FPGA:68 416 个 LE,250 个 M4K RAM 块,总计 1 152 000 个 RAM 位,150 个片内乘法器,4 个 PLL,622 个用户 I/O 引脚,896 个引脚,FineLine BGA 封装。

(2) 配置芯片和 USB Blaster:配置芯片为 EPCS16,用于编程和 API 控制的 USB Blaster,支持 JTAG 和 AS 两种编程模式。

(3) SSRAM:2MB(512K×36b)标准同步 SRAM,可用于 Nios Ⅱ 处理器的存储器

或直接由 FPGA 控制使用。

(4) SDRAM：有 2 个 32MB(4M×16b×4banks) 单数据率同步动态 RAM，可用于 Nios Ⅱ处理器的存储器或直接由 FPGA 控制使用。

(5) Flash memory：8MB NOR Flash 存储器，支持字节和字两种模式，可用于 Nios Ⅱ处理器的存储器或直接由 FPGA 控制使用。

(6) SD 卡插座：支持 SPI 和 1-bit SD 模式，可用于 Nios Ⅱ处理器的存储器。

(7) 按钮开关：4 个按钮开关，已经由施密特触发器消抖，通常为高，按下时输出一个低电平。

(8) 拨动开关：18 个用于输入的拨动开关，拨到上面为'1'，拨到下面为'0'。

(9) Clock inputs：有 50MHz 和 28.63MHz 振荡器和 SMA 外部时钟输入。

(10) Audio CODEC：Wolfson WM8731 24b sigma-delta 音频 CODEC，带有 line-in、line-out 和 microphone-in 接口，采样频率 8～96kHz，用于各种音频输入输出。

(11) VGA output：ADV7123 140MHz triple 10b high-speed video DAC，15 引脚高密 D-sub 接口，支持 1600×1200 at 100Hz 刷新频率，可作 FPGA 的输出显示用。

(12) NTSC/PAL/SECAM TV 解码电路：用了 2 个 ADV7180 Multi-format SDTV Video Decoders，用于各种视频输入输出。

(13) 10/100 以太网控制器：带有通用处理器接口的集成 MAC 和 PHY，支持 100Base-T 和 10Base-T 应用，带有 auto-MDIX，支持 10Mbps 和 100Mbps 全双工操作，完全兼容 IEEE 802.3u 规范，支持 IP/TCP/UDP 求和校验半双工模式背压流控。

(14) USB 主/从控制器：满足国际串行总线标准 Rev. 2.0，支持全速和低速数据传送。支持主/从两个 USB 口(A 口是主口，B 口是从口)，支持大部分处理器的高速并行接口，支持 PIO 和 DMA 操作。

(15) 串行接口：一个 RS-232(DB-9 串行接口)接口，一个用于接鼠标或键盘的 PS/2 接口。

(16) IrDA 转换器：含有一个 115.2kbps 的红外转换器，32mA LED 驱动电流，集成 EMI 保护，满足 IEC825-1 标准，边缘输入检测。

(17) 两个 40 引脚的扩展头：Cyclone Ⅱ的 72 个 I/O 引脚，和 8 个电源端和地端接到两个 40 引脚的扩展头上，有二极管和电阻保护。

1.3.2 USB-Blaster 的驱动安装

除了加电之外，为了让 DE2-70 开发板/教学实验板正常进行开发工作，还需要在主机上安装 USB-Blaster 电缆的驱动程序以支持 PC 端的开发软件，例如 Quartus Ⅱ、Nios Ⅱ IDE 等。

以下是 USB-Blaster 驱动程序在 Windows XP 下的安装步骤。

安装环境相关说明如下。

开发软件：Quartus Ⅱ 10.0 以上。

开发平台：Windows XP SP3/Windows 7。

将 DE2-70 实验平台的 USB-Blaster 接口(开发板上部最左边)和 USB 连接线接好，

接线的另一头插入主机的 USB 接口，Windows XP 发现新硬件后会弹出一个对话框，如图 1-18 所示。选择“从列表或指定位置安装”，单击“下一步”按钮。



图 1-18　找到新的硬件向导

选择搜索和安装选项，USB-Blaster 的驱动程序在 Quartus Ⅱ 的安装目录下，请指定 USB-Blaster 的驱动程序路径，如图 1-19 所示。单击“下一步”按钮，继续安装。

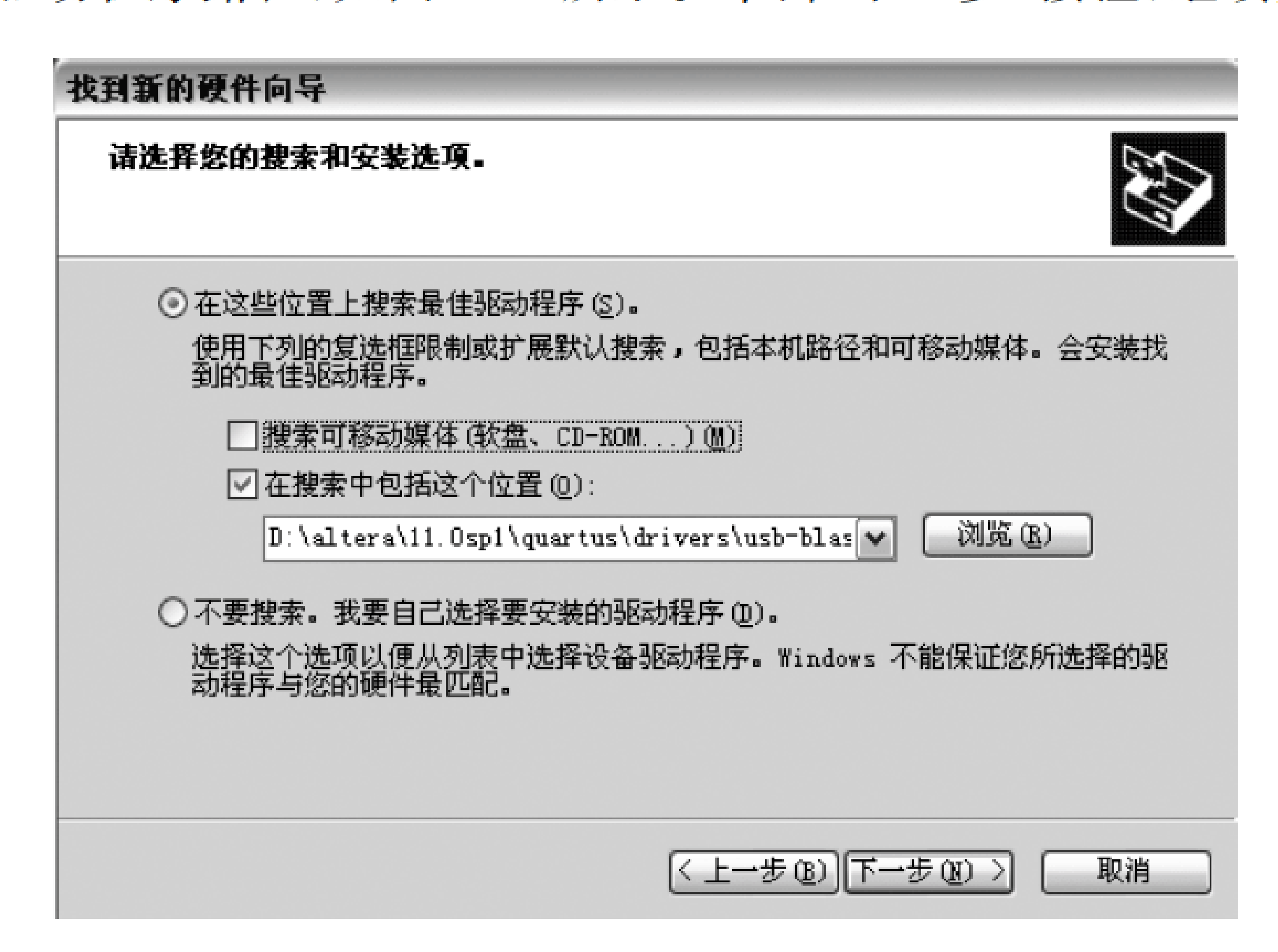


图 1-19　选择搜索和安装选项

选定安装程序的目录后，单击“下一步”按钮，继续安装 USB-Blaster 的驱动程序，如图 1-20 所示。

完成安装的提示对话框如图 1-21 所示。

右键单击“我的电脑”图标，进入属性页，再进入“硬件”选项卡，单击“设备管理器”对话框，单击“通用串行总线控制器”图标，查看安装是否成功，如图 1-22 所示。

以上就是 DE2-70 实验平台的驱动安装过程。



图 1-20　安装 USB-Blaster 的驱动程序



图 1-21　驱动程序安装完成



图 1-22　驱动程序安装成功与否检查

1.3.3 DE2-70开发板的使用

本节将介绍DE2-70开发板上各个外围芯片以及每个芯片和FPGA之间的I/O口引脚连接情况。

在DE2-70开发板上有一个专门用于存储Cyclone Ⅱ FPGA配置信息的串行EEPROM芯片。这些配置信息在每次开发板加电时自动从EEPROM加载到FPGA中。通过使用Quartus Ⅱ软件,可以在任意时刻重新对FPGA进行编程配置,也可以改变存储在EEPROM芯片中非易失的配置信息。下面将介绍对FPGA的两种编程方法。

(1) JTAG(Joint Test Action Group)编程:满足IEEE标准,在这种编程方式中,配置数据直接下载到Cyclone Ⅱ FPGA芯片中。只要保持开发板不掉电,FPGA就会保留住这些配置数据;而一旦板子掉电这些配置数据就会丢失。

(2) AS编程(Active Serial programming):在这种编程方式下,FPGA的配置数据被下载到Altera EPCS16串行EEPROM芯片中。EEPROM是一种非易失性存储器,即使DE2-70板子掉电,这些配置数据也会被保留。一旦开发板加电,这些在芯片EPCS16中的配置数据就自动加载到Cyclone Ⅱ FPGA中。

下面分别介绍JTAG和AS编程的步骤。对两种编程方式而言,DE2-70开发板都是通过一根USB数据线与主机相连。使用这种连接时,开发板会被主机识别成一个Altera USB Blaster设备。

1. 以JTAG模式配置FPGA

图1-23说明了如何用JTAG方式下载数据到FPGA中,用这种方式配置FPGA要完成以下步骤:

(1) 确认DE2-70开发板已经加电。

(2) 将开发板自带的USB数据线与DE2-70开发板上的USB Blaster端口相连,如图1-23所示。

(3) 把RUN/PROG开关(在板子左边)拨向RUN位置,选用JTAG方式配置FPGA。

(4) 通过Quartus Ⅱ软件工具条上的Programmer选项,选择.sof文件对开发板上的FPGA进行编程。

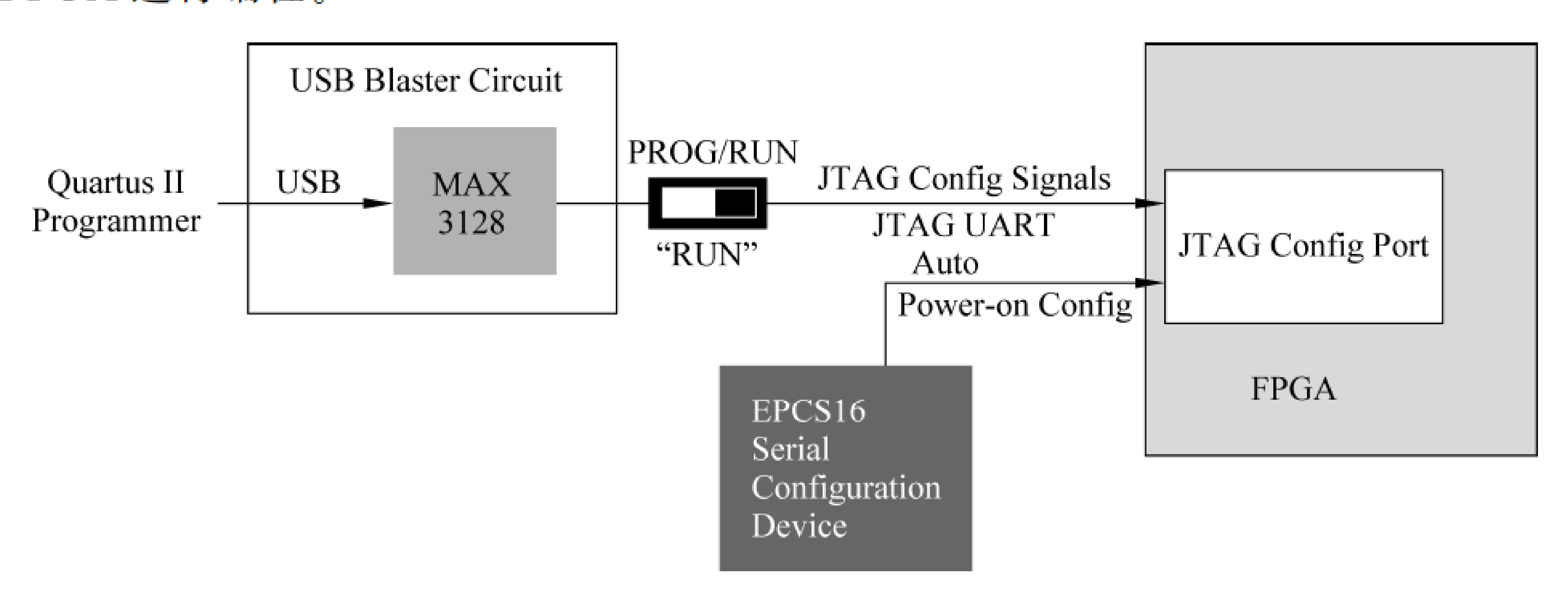


图1-23 JTAG配置示意图

2. 以 AS 模式配置 FPGA

图 1-24 介绍的是 AS 编程模式。请按以下步骤完成对 FPGA 的 AS 模式编程：

(1) 确认 DE2-70 开发板已经加电。

(2) 将开发板自带的 USB 数据线与 DE2-70 开发板上的 USB Blaster 端口相连，如图 1-24 所示。

(3) 通过把 RUN/PROG 开关(在板子左边)拨向 PROG 位置，选择以 AS 方式配置 FPGA。

(4) 通过 Quartus Ⅱ 软件工具条上的 Programmer 选项，选择.pof 文件对开发板上的 FPGA 配置芯片 EPCS16 进行编程。

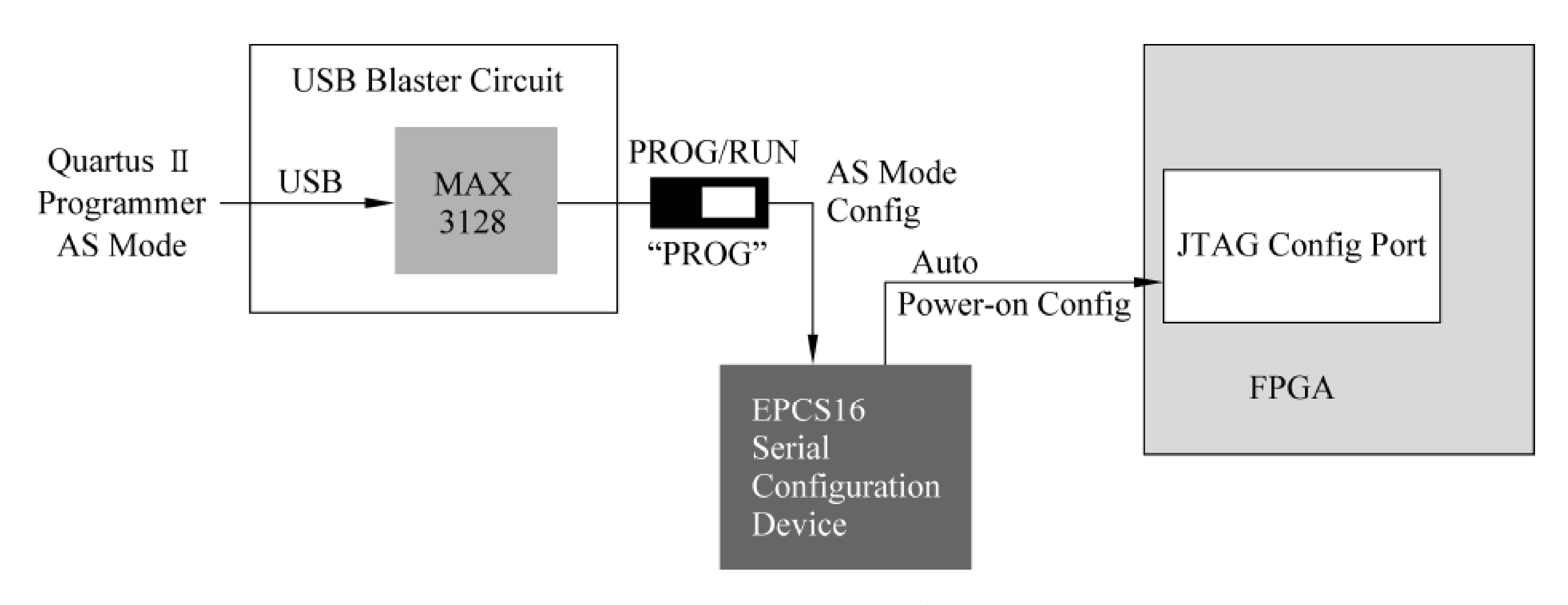


图 1-24 AS 配置示意图

配置完成后，把 RUN/PROG 开关拨回 RUN 位置，给开发板重新加电。这时，刚才配置在 EPCS16 中的数据被加载进 FPGA 芯片中。

另外，对 JTAG 和 AS 编程的使用而言，DE2-70 开发板上的 USB Blaster 端口也可以用来从主机远程控制板子的一些特性。

第 2 章 EDA 技术基础知识

EDA(Electronic Design Automation)即电子设计自动化,是计算机辅助设计(CAD)的一种。EDA 技术就是设计者以计算机为工具,在 EDA 软件平台上,用硬件描述语言(HDL)或者电路原理图的方式完成设计文件,然后由计算机自动地完成逻辑编译、综合、优化、布局、布线和仿真,直至对目标芯片(如 CPLD、FPGA)进行配置、映射和编程下载等工作。EDA 技术极大地提高了电路设计的效率和可重复性,减轻了设计者的劳动强度。

本章 2.1 节首先介绍数字系统设计的过程。2.2 节介绍利用 Altera 公司的 EDA 设计软件 Quartus Ⅱ 开发 FPGA 的常用步骤。2.3 节简单介绍常用的硬件描述语言 Verilog HDL,列举两个用 Verilog 硬件描述语言设计电路的实例,由此可以看出 Verilog HDL 的概貌,从而有助于进一步了解硬件描述语言设计电路的过程。

2.1 数字逻辑系统设计过程

无论是一个简单的多路开关控制灯实验还是一个复杂的计算机系统,数字系统的设计都可以按照如图 2-1 所示的统一的设计流程来实现。

项目说明书由用户提供给设计者,说明书详细描述了用户要求的产品的规格和性能,设计者必须详细深入地阅读项目说明书,根据说明书来定义将要设计的数字系统的电路行为和电路特性,并由此完成电路的总体设计工作。

对于一个数字系统,一般很难在一个电路模块中完成所有的功能,这时设计者就需要设计数个相对独立的功能模块,每个模块分别完成产品的某个特定功能。设计者分别设计各个模块:定义各个模块的电路、对各个电路模块进行仿真。对于一个小的数字系统,所有这些模块可能是由一个设计者独立完成的,而对于一个大型系统,就需要一个项目管理者(项目经理)将任务分成几个模块,每个模块分配给一个单独的设计者(工程师)来完成。

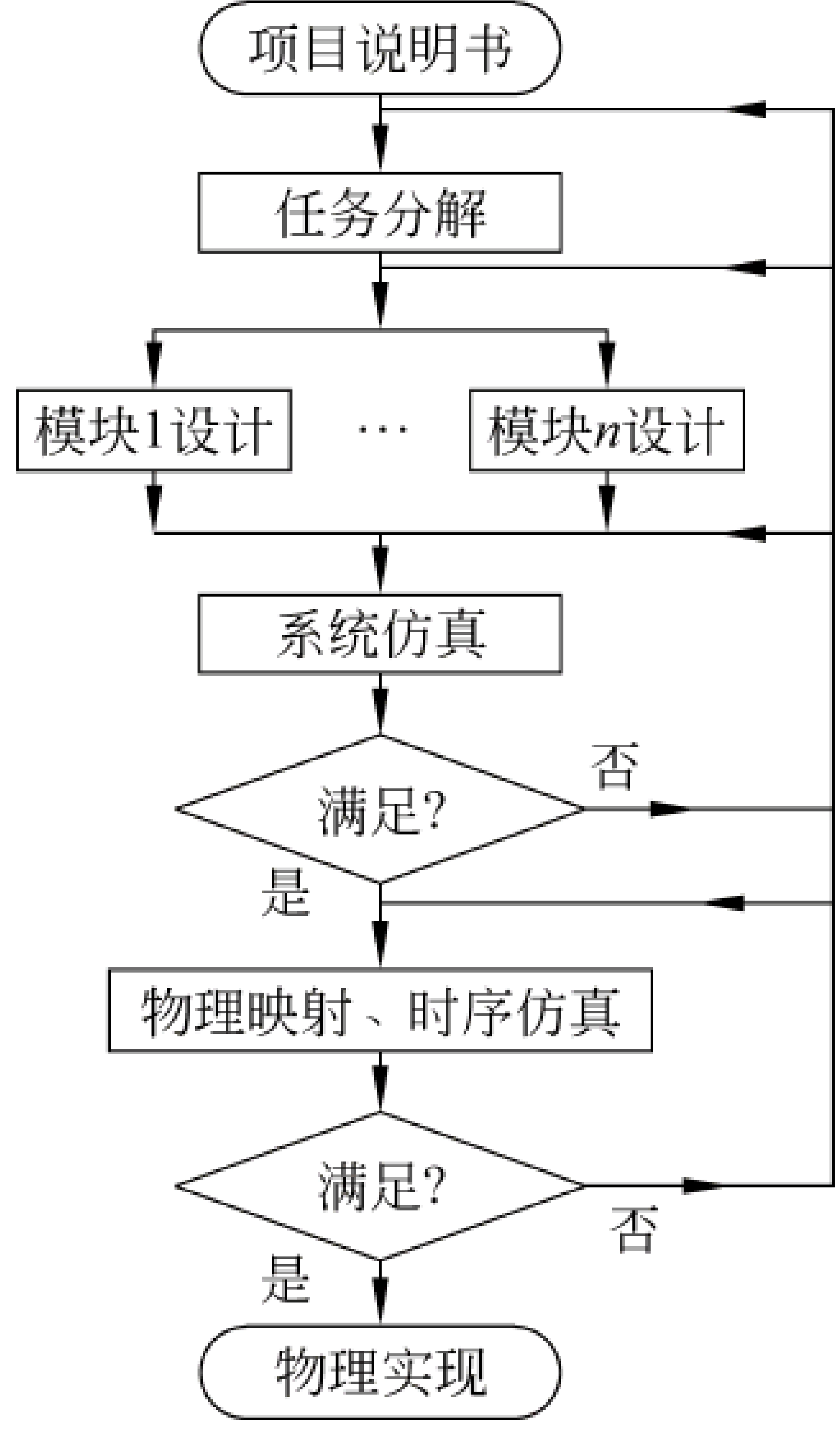


图 2-1 数字系统的设计流程

系统仿真是在各个模块都设计完成后，将每个模块连接成为一个完整的数字系统，然后对整个系统进行功能仿真的过程，系统仿真过程中可能出现很多错误或者不完全符合项目说明书的情况，这时就要修改错误，修改错误可能涉及上一步设计过程的各个环节，例如，修改系统的连接、修改各个小模块内部设计甚至修改模块功能的划分等。修改设计要循序渐进，直至功能仿真能满足项目说明书的所有功能和要求。

如果功能仿真正确，就意味着所设计的电路在理论上能实现用户要求的功能。理论电路要在实际的印刷电路板上才能实现其功能，物理映射就是将理论上的电路映射到实际的电路板上，物理映射的过程是将每个功能芯片映射到数字电路板上的具体位置，以及设计功能芯片之间的连线方案等。物理映射后需要进行时序仿真，时序仿真是测试数字系统在实际电路板上运行的速度和性能是否满足用户的需要，如果不能满足，首先调整电路的物理映射，如果仅调整物理映射还不能达到项目说明书的要求，那么就需要回到理论设计阶段去修改和完善设计了。

时序仿真完成后，数字系统就可以做成成品了，这时数字系统的设计工作基本完成，设计者可以将设计好的电路直接交由专门的厂家来完成物理实现。物理实现的过程中大部分的问题可以通过修改物理布线来修正，但是也有可能发生必须修改设计的大问题。

大型的数字系统都是由一个个相对独立的功能模块组成的，目前我们学习的内容一般也仅限于设计某个功能相对单一的模块或者小型的数字系统，对于只涉及单一功能模块的小型数字系统的设计，其基本设计过程如图 2-2 所示。

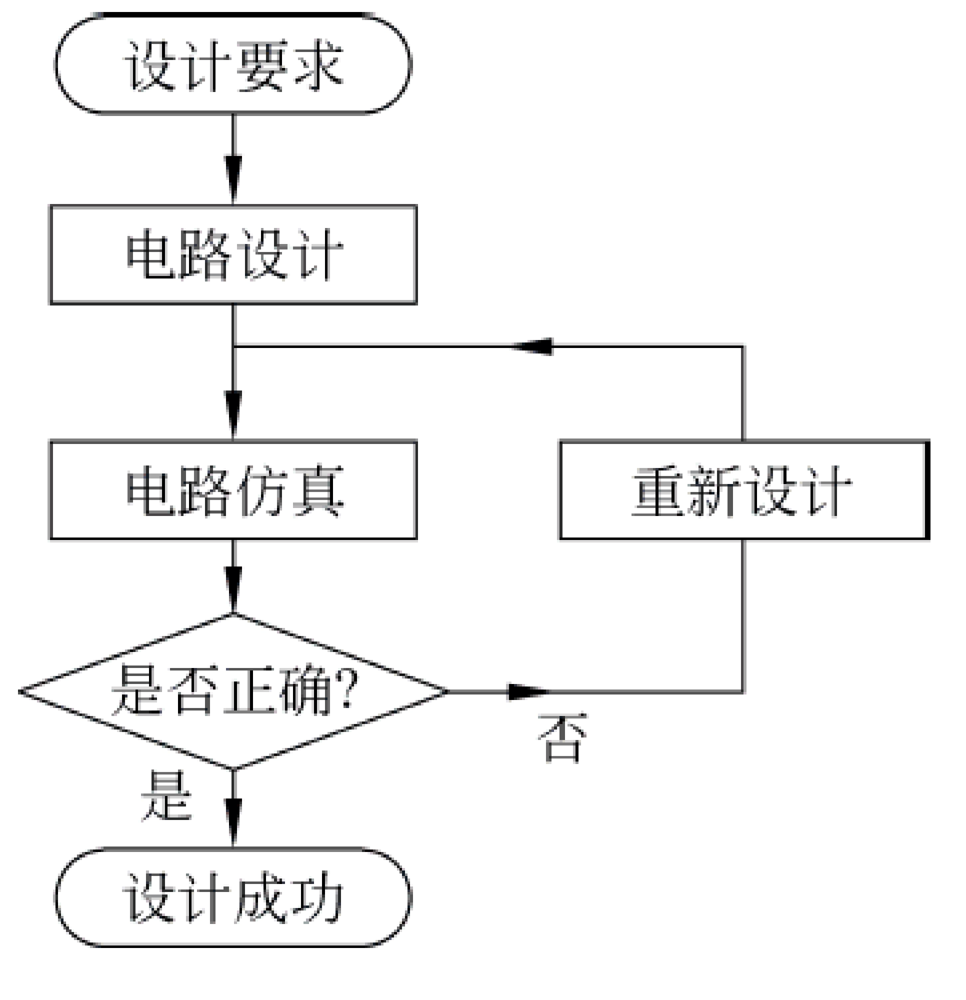


图 2-2　基本单元设计过程

设计要求是关于数字电路的要求、特性和功能等的详细描述。设计者的工作就是根据设计要求，逐步完成功能电路。

电路设计的工作是电路系统的初步设计，这一步的工作主要如下：

(1) 分析系统功能，总结出系统要完成的工作。

(2) 对实际问题进行数字抽象，确定数字系统的输入信号和输出信号等。

(3) 建立数字模型，描述出数字系统输入信号和输出信号之间的关系，最好能够列出系统的真值表。

(4) 对数字模型进行化简，利用逻辑表达式、卡诺图或者其他方式将电路化简，使电路易于实现。

(5) 设计具体电路。

这些步骤都需要设计者根据自己的经验、知识、技术和直觉等来设计，这一步工作一般需要设计者手工完成。

电路仿真的过程是利用优秀的 CAD 工具，来描述上一步已由设计者手工完成的电路设计，验证手工设计的电路是否满足项目要求，如果不满足，就需要重新设计，直至设计成功。每个可编程逻辑芯片的生产厂家都会为自己的产品开发专用的设计软件，例如用 Xilinx 公司的 FPGA 来实现硬件电路需要使用编译软件 ISE，而我们的实验平台选用的

FPGA 是 Altera 公司生成的 FPGA，因此我们的设计软件选用的也是 Altera 公司的 Quartus Ⅱ设计软件。

2.2 Quartus Ⅱ使用入门

Quartus Ⅱ 软件是 Altera 公司提供的完整的 EDA 设计工具，是数字逻辑系统设计的仿真工具，它对设计者的设计进行编译、仿真、模拟，最后生成用于配置 FPGA 的文件，并将其配置到 FPGA 开发平台上，实现所设计的电路。Quartus Ⅱ软件拥有 FPGA 和 CPLD 设计的所有阶段的解决方案，我们可以使用 Quartus Ⅱ 软件完成数字逻辑设计仿真的所有阶段工作流程。有关 Quartus Ⅱ设计流程的图示说明，请参见图 2-3。

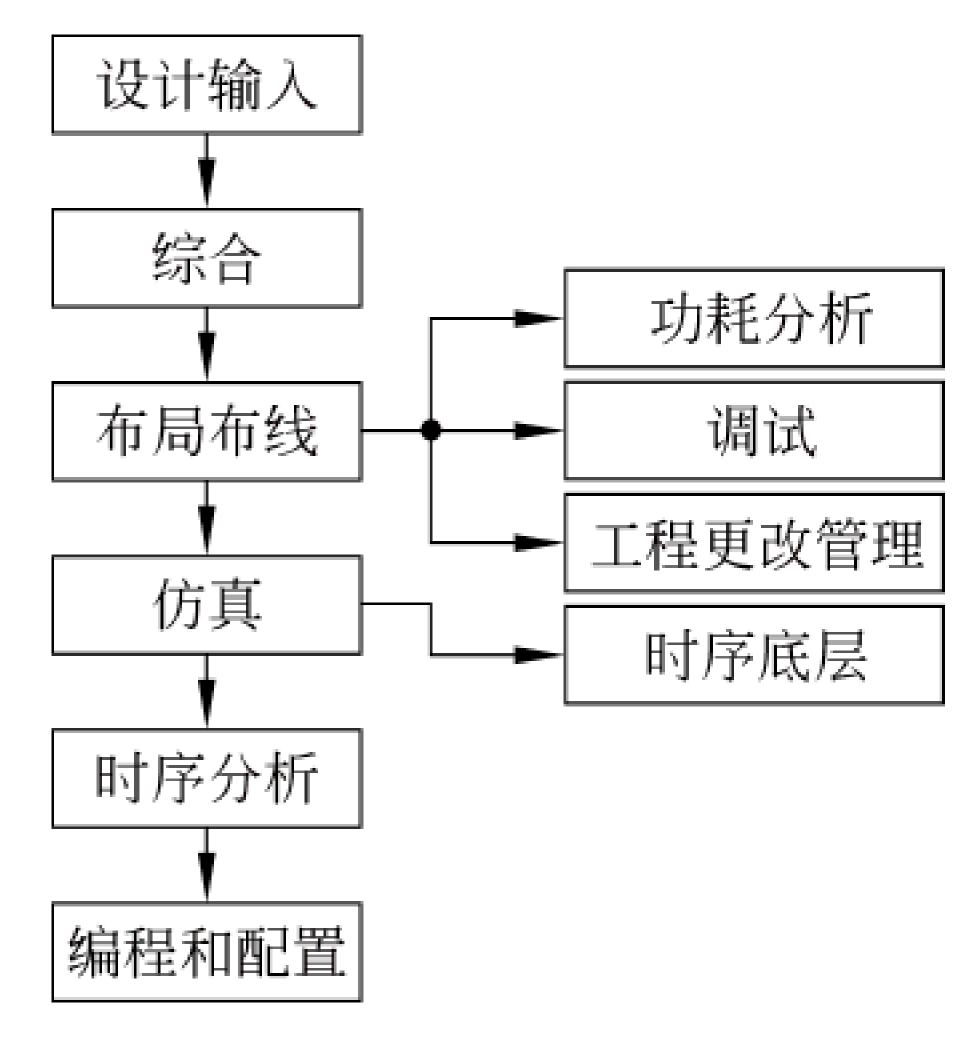


图 2-3 Quartus Ⅱ的设计流程图

图 2-4 显示了 Quartus Ⅱ图形用户界面为设计流程的每个阶段所提供的功能。

本节利用一个具体的实验来设计一个简单的组合逻辑电路，详细介绍一个具体的数字逻辑电路的设计过程，并利用 Quartus Ⅱ 软件对其进行仿真，最后下载到 FPGA 开发平台上进行验证。通过本次实验，可以了解数字逻辑设计的方法和过程，并初步学习 Quartus Ⅱ 软件使用方法。

221 问题分析和设计

简单的数字逻辑电路设计中最常用的方法，如问题抽象、列真值表、化简等内容必须要设计者自己手工实现，这是在数字逻辑电路理论课中所讨论的内容，这里不加详述。我们只简单介绍一个具体的数字逻辑电路从问题的提出到电路实现的过程。

2.2.1.1 问题描述

一间很大的屋子有三个门，每个门边有一个开关来控制大房间里仅有的一个灯，要求设计一个电路，使得无论人从哪个门进入或者走出此房间，只要改变此门边的开关状态就可以开灯或者关灯，我们姑且称之为“三端控制灯”。

2.2.1.2 分析问题，确定输入输出端

本实验要求三个开关可以改变同一个灯的状态，那么此电路应该有三个输入端(三个开关)和一个输出端(灯)。

2.2.1.3 分析输入信号和输出信号的关系列出真值表

假设三个开关分别为 A、B 和 C，开关的“开”、“关”状态分别用“1”和“0”表示；灯为 F，灯的“亮”和“灭”状态也分别用“1”和“0”表示。列出本实验的真值表如表 2-1 所示。

设计输入
文本编辑器
块与符号编辑器
MegaWizard插件管理器
编辑器
布局图编辑器

综合
分析和综合
VHDL、Verilog HDL与AHDL
辅助设计
RTL查看器
Technology Map Viewer
增量综合

布局布线
Fitter
分配编辑器
布局图编辑器
芯片编辑器
报告窗口
递增配合
Design Space Explorer

时序分析
时序分析器
报告窗口
Technology Map Viewer

仿真
仿真器
波形编辑器

编程和配置
汇编器
Programmer
转换编程文件

系统级设计
SOPC Builder
DSP Builder

软件开发
Software Builder

基于块的设计
LogicLock 窗口
布局图编辑器
VQM Writer

EDA界面
EDA网表编写程序

Power Analysis
PowerPlay Analyzer 工具
PowerPlay Early Power Estimator

时序底层
布局图编辑器
LogicLock 窗口
Timing Optimization Advisor
Design Space Explorer

调试
SignalTap II
SignalProbe
In-System Memory Content Editor
RTL 查看器
Technology Map Viewer
芯片编辑器

工程更改管理
工程更改管理
资源属性编辑器
更改管理器

图 2-4　图形用户界面的功能

表 2-1　三端控制灯真值表

A	*B*	*C*	*F*	*A*	*B*	*C*	*F*
0	0	0	0	1	0	0	1
0	0	1	1	1	0	1	0
0	1	0	1	1	1	0	0
0	1	1	0	1	1	1	1

2.2.1.4 根据真值表写出逻辑表达式或者卡诺图，并进行化简

根据真值表，写出本电路的逻辑表达式为：

$$F=\overline{A}\overline{B}C+\overline{A}B\overline{C}+A\overline{B}\overline{C}+ABC$$

该表达式已不能再化简为更简单的表达式。

或者根据真值表，画出本电路的卡诺图，如图 2-5 所示。

由卡诺图得出表达式为：

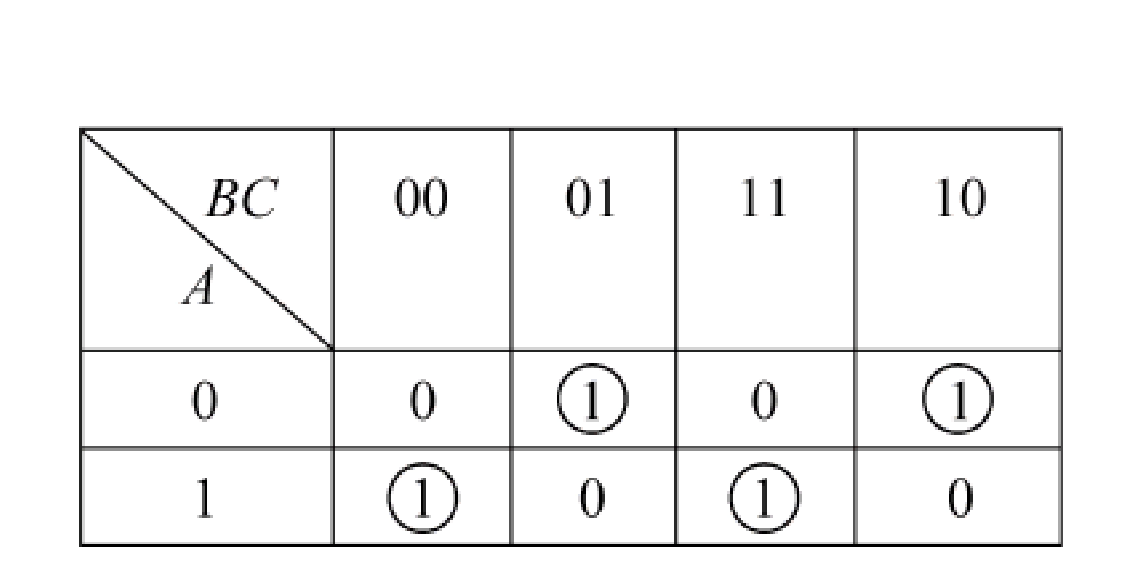
$$F=\overline{A}\overline{B}C+\overline{A}B\overline{C}+A\overline{B}\overline{C}+ABC$$

2.2.1.5 根据化简的逻辑表达式，画出电路图

由上述分析得出本电路的表达式为：

$$F=\overline{A}\overline{B}C+\overline{A}B\overline{C}+A\overline{B}\overline{C}+ABC$$

根据表达式画出电路图，如图 2-6 所示。



A \ BC	00	01	11	10
0	0	①	0	①
1	①	0	①	0

图 2-5 三端控制灯的卡诺图

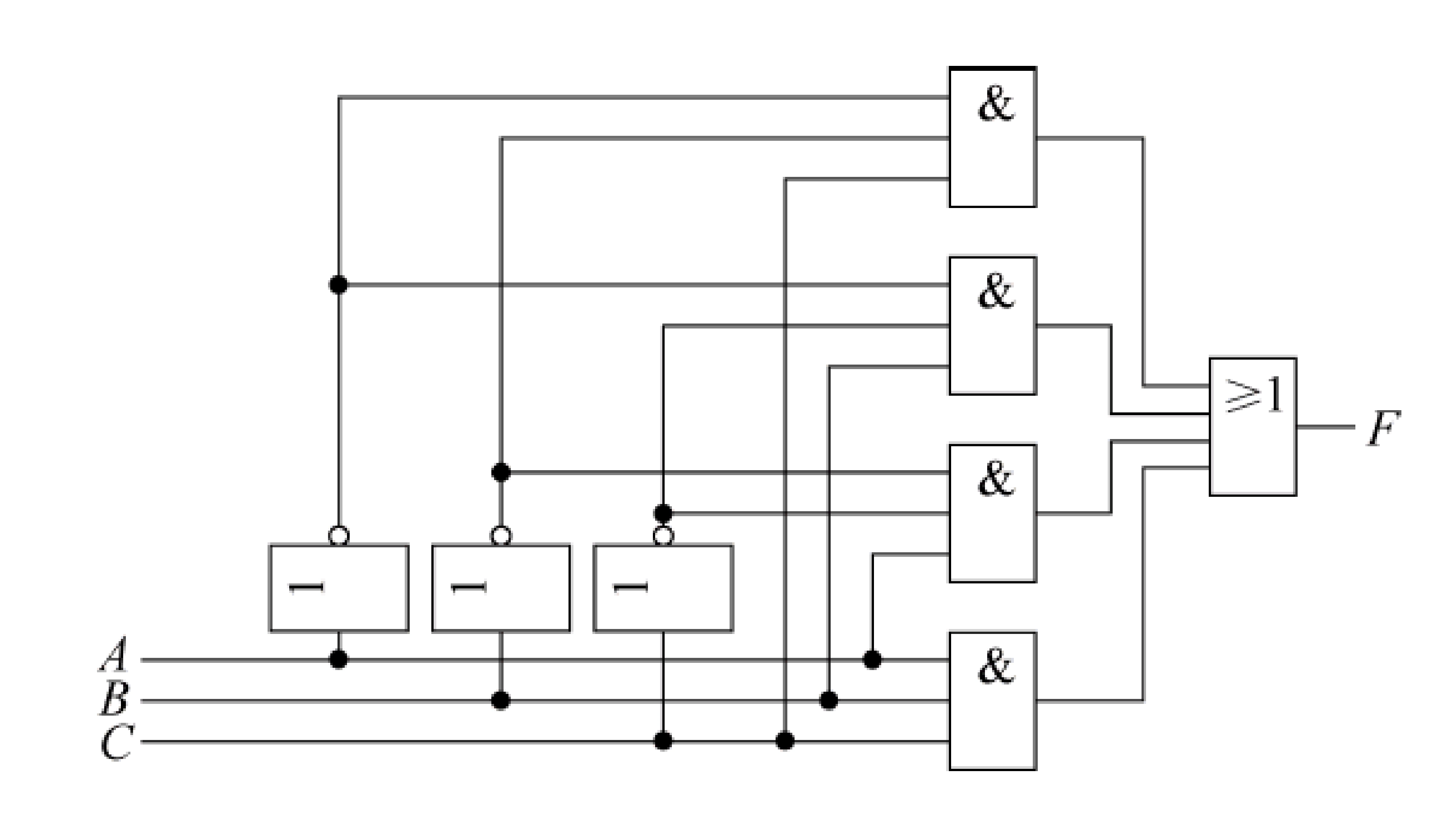


图 2-6 三端控制灯电路图

222 利用 Quartus Ⅱ 完成电路仿真

2.2.2.1 建立 Quartus 工程

在利用 Quartus Ⅱ进行设计仿真之初，需要为设计的项目建立一个工程，工程包括设计和编译过程中所有的文件：软件源文件和编译过程中产生的其他过程文件。

选择"开始→程序→Altera 12.0→Quartus Ⅱ 12.0sp2 Web Edition→Quartus Ⅱ 12.0sp2 Web Edition"，或者双击桌面上的图标，启动软件，打开 Quartus Ⅱ工作环境。第一次打开 Quartus Ⅱ工作环境，会出现要求对其认证的对话框，如图 2-7 所示。

Quartus Ⅱ使用时需要对其进行认证，新安装 Quartus Ⅱ软件的用户可以有 30 天的免费试用期，在此期间不用进行认证，30 天后必须进行认证，没有经过认证的 Quartus Ⅱ在编译时不能产生用于配置 FPGA 的文件(.sof 文件或者.pof 文件)。没有经过认证的 Quartus Ⅱ软件在每次启动时都会询问用户是启动 30 天免费试用版本，还是选择进行 license 认证。如果读者有 Altera 官方提供的 license 认证文件，请选择"If you have a valid license file, specify the location of your license file"选项。

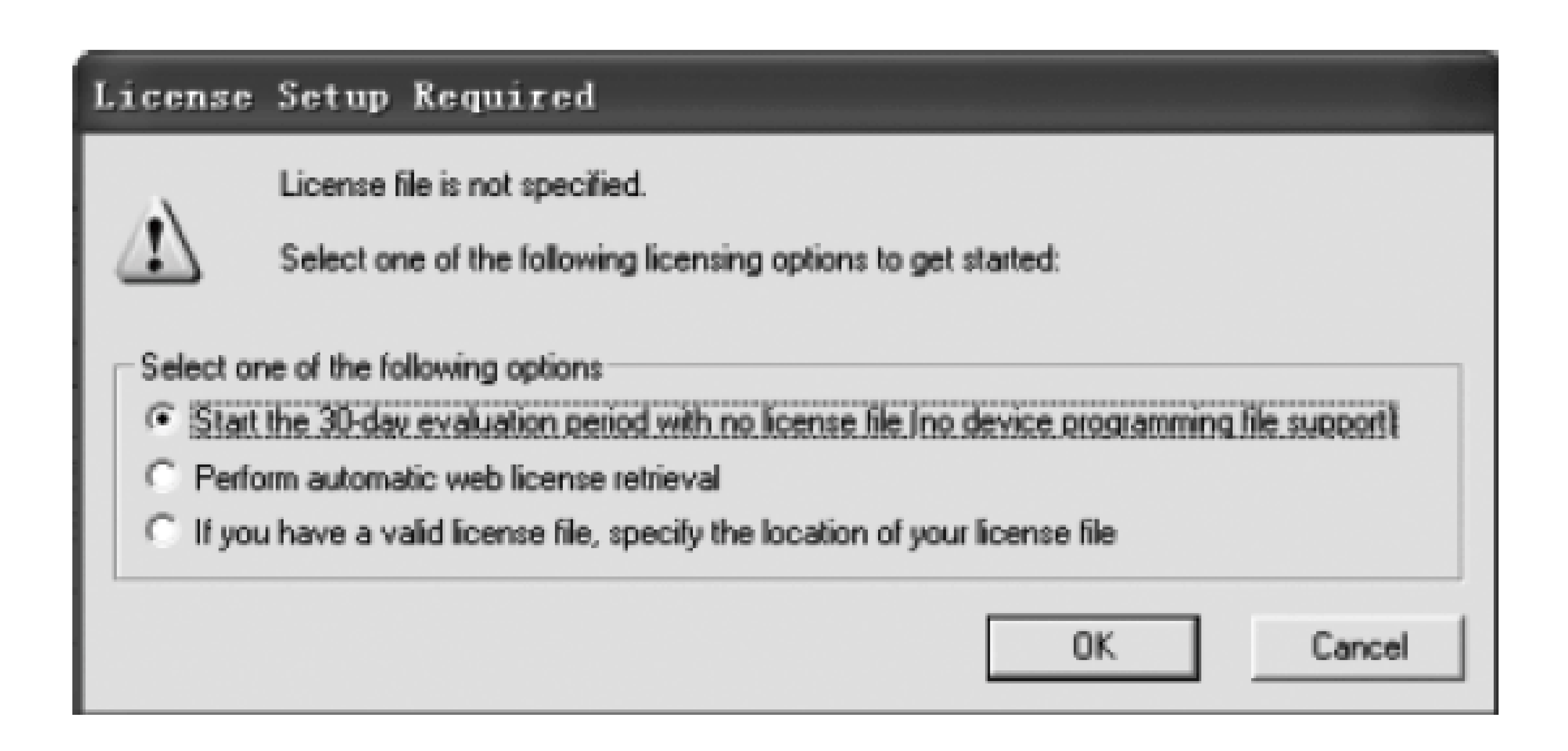


图 2-7　license 选择

选择有可使用的 license 文件后，系统会弹出一个选择此 license 文件路径的提示框，请将你的 license 文件的存放路径添加进去。学校或企业的 license 文件一般保存在服务器上，请在实验的时候确信你的机器能够正确地连到服务器上，单击 Quartus Ⅱ 工作界面工具栏中的“Tools→License Setup”，弹出如图 2-8 所示的对话框，在 License file 栏中填入正确的 license 文件地址，单击“OK”按钮进行认证。

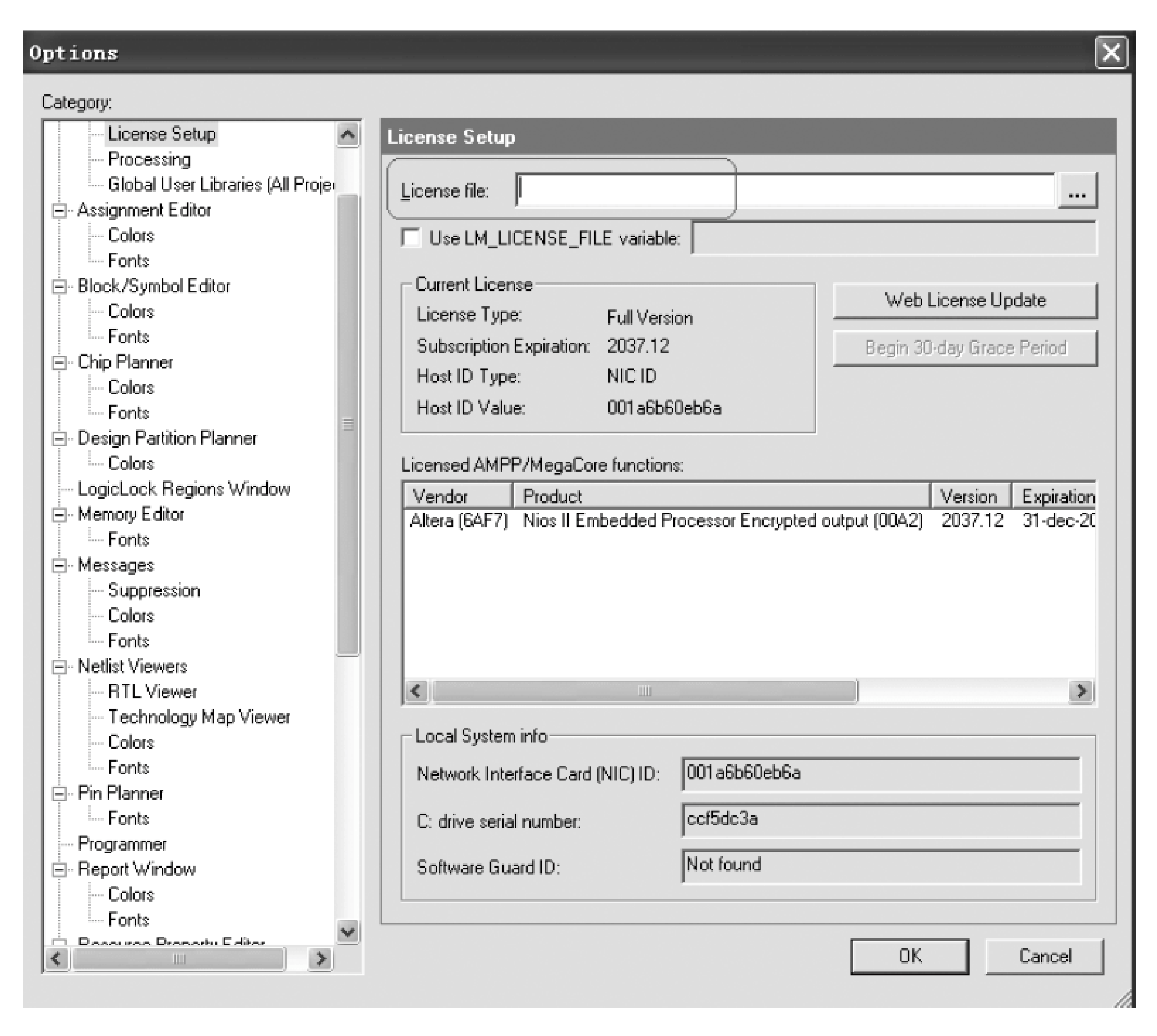


图 2-8　选择认证文件

认证完成后，进入 Quartus Ⅱ 的工作界面，如图 2-9 所示。用户可以根据自己的需要打开相应项目，也可以直接进入工作界面。

单击菜单项“File→New Project Wizard”，为自己的设计建立一个新项目并指定目标器件或器件系列，如图 2-10 所示。

打开 Wizard 后，将出现显示建立新项目的介绍框，介绍在新建项目过程中所要完成的具体工作，如图 2-11 所示。

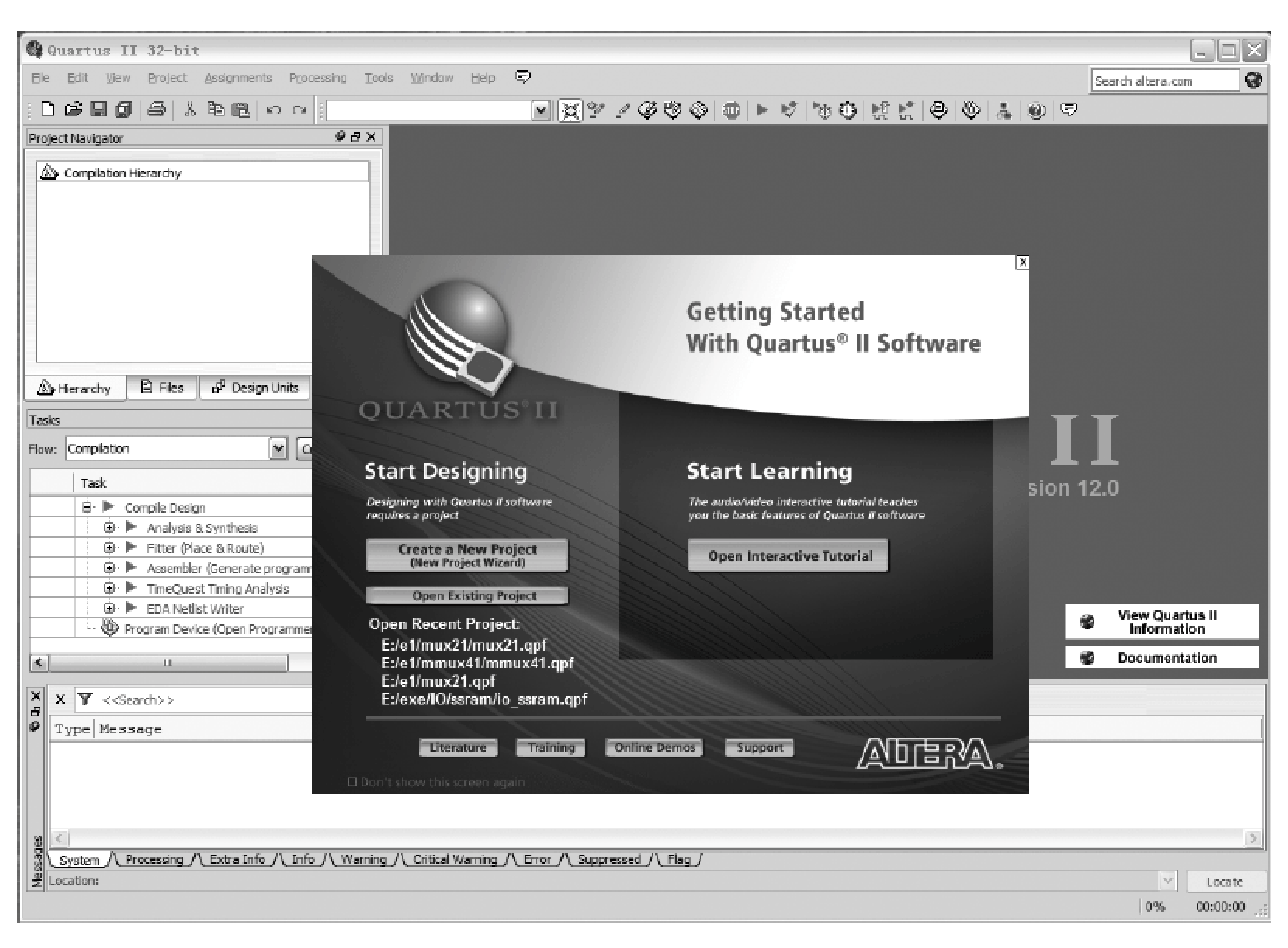


图 2-9　打开 Quartus Ⅱ工作环境

图 2-10　建立新项目

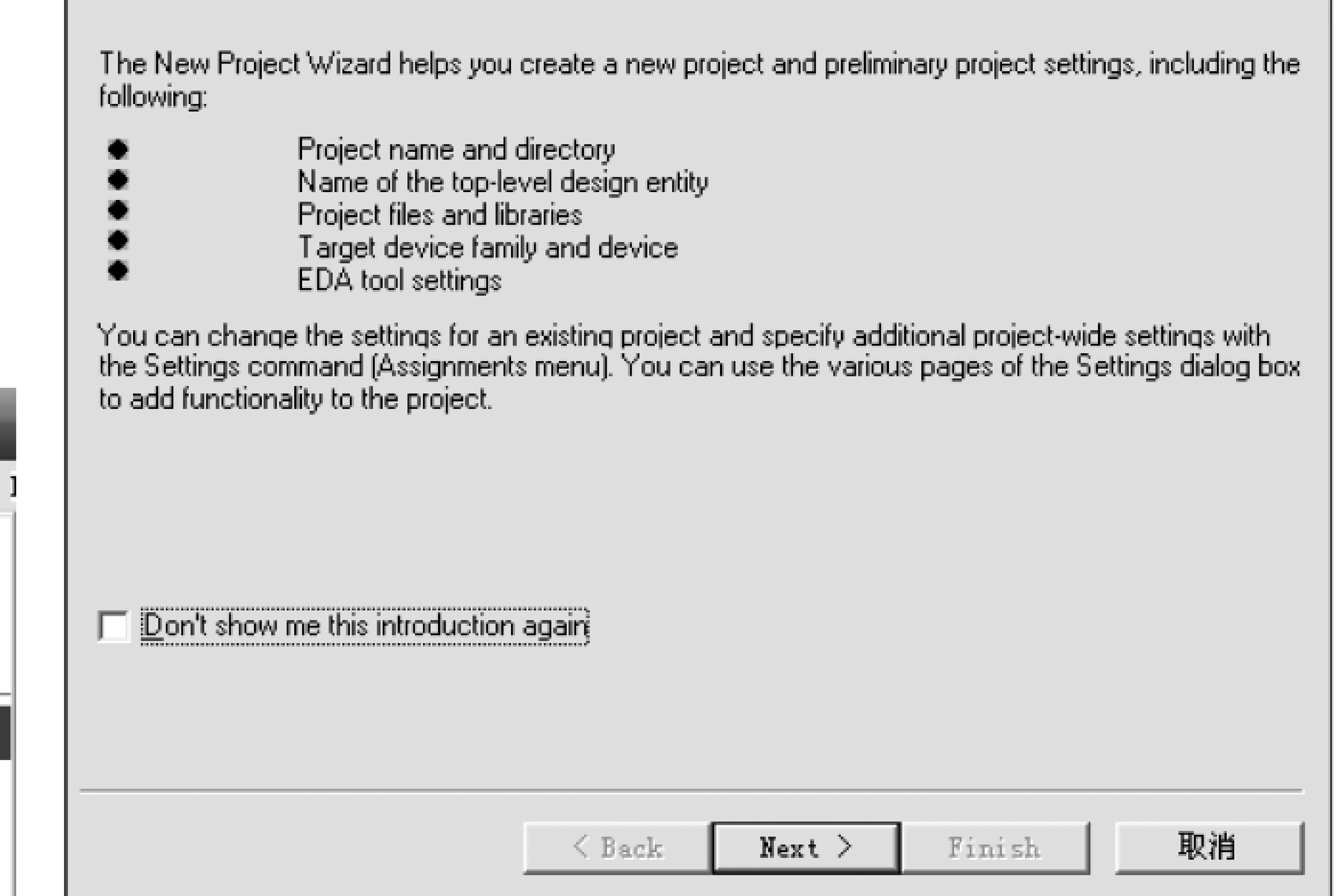


图 2-11　新建项目工作介绍

单击“Next”按钮，进入工程路径的选择和工程名称的设定，如图 2-12 所示。

在第一行是工程路径的选择，默认选择的是 Quartus Ⅱ的安装路径，请重新指定工程的路径。

注意：请不要将自己的设计文件直接存放在系统默认的安装目录下，更不要将工程文件直接放在安装目录中，建议在自己的计算机中专门指定一个工作目录/路径，存放自己的所有工程。

在第二行输入工程名称，建议每一个工程都在自己的工作目录下新建一个文件夹，用于保存此工程的所有文件。

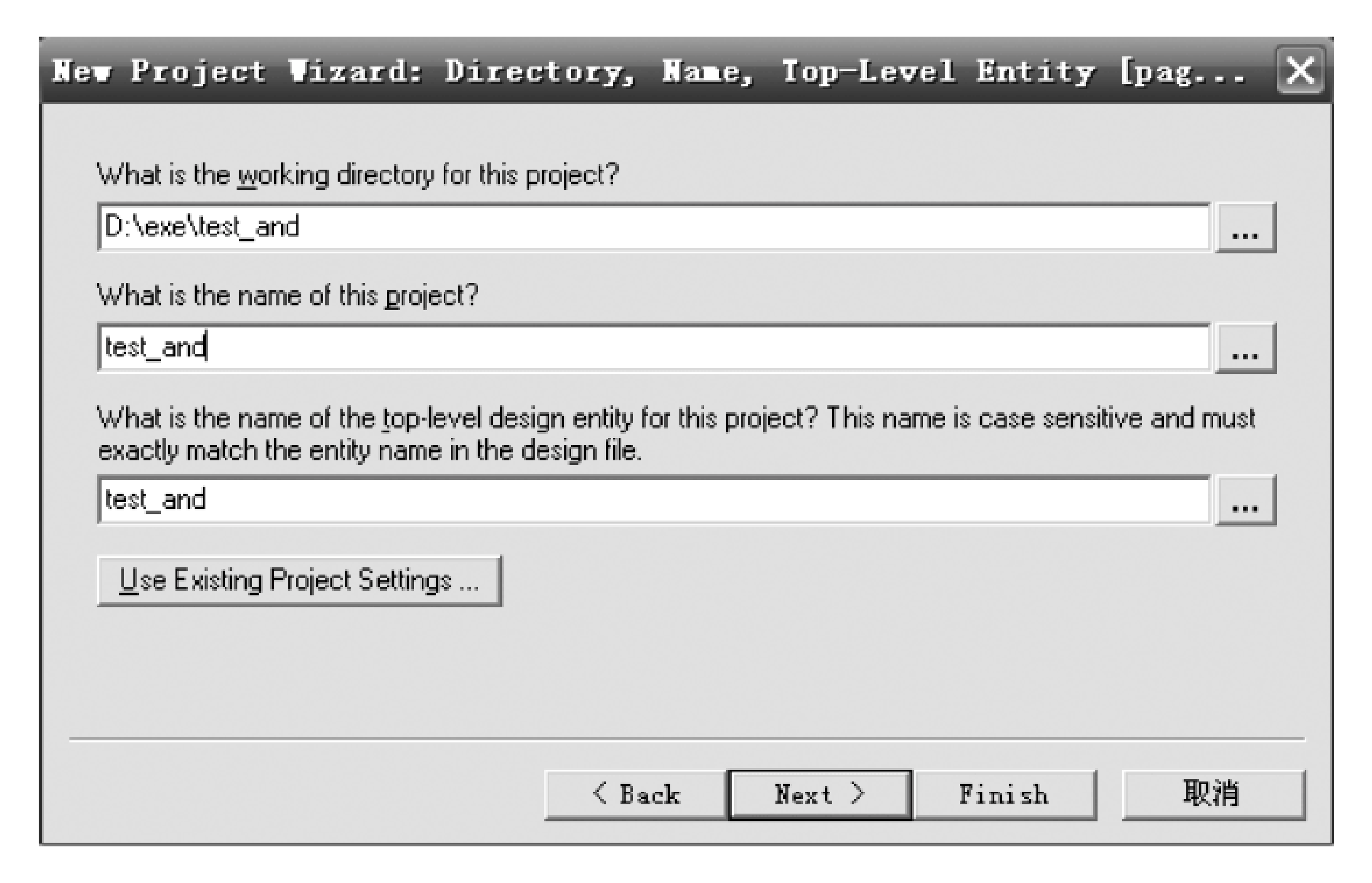


图 2-12　顶层实体名默认同于项目名

在第三行输入工程的顶层实体名，这里输入的顶层实体名必须与之后设计文件的顶层实体名相同，默认的顶层实体名与工程名相同，用户也可以根据需要输入不同的顶层实体名。

注意： 工程名和顶层实体名可以根据自己的习惯命名，但必须是英文字母开头，英文字母、数字和下划线的组合。工程文件夹所在的路径名和文件夹，不能用中文命名，不能用空格和括号，也不能以数字开头。

添加设计文件。界面如图 2-13 所示。如果用户之前已经有设计好的文件，并且要在此工程用到这个文件(例如 HDL 语言文件或原理图文件)，那么可在此时将文件添加到工程中。如果没有完成的设计文件则单击“Next”按钮继续。

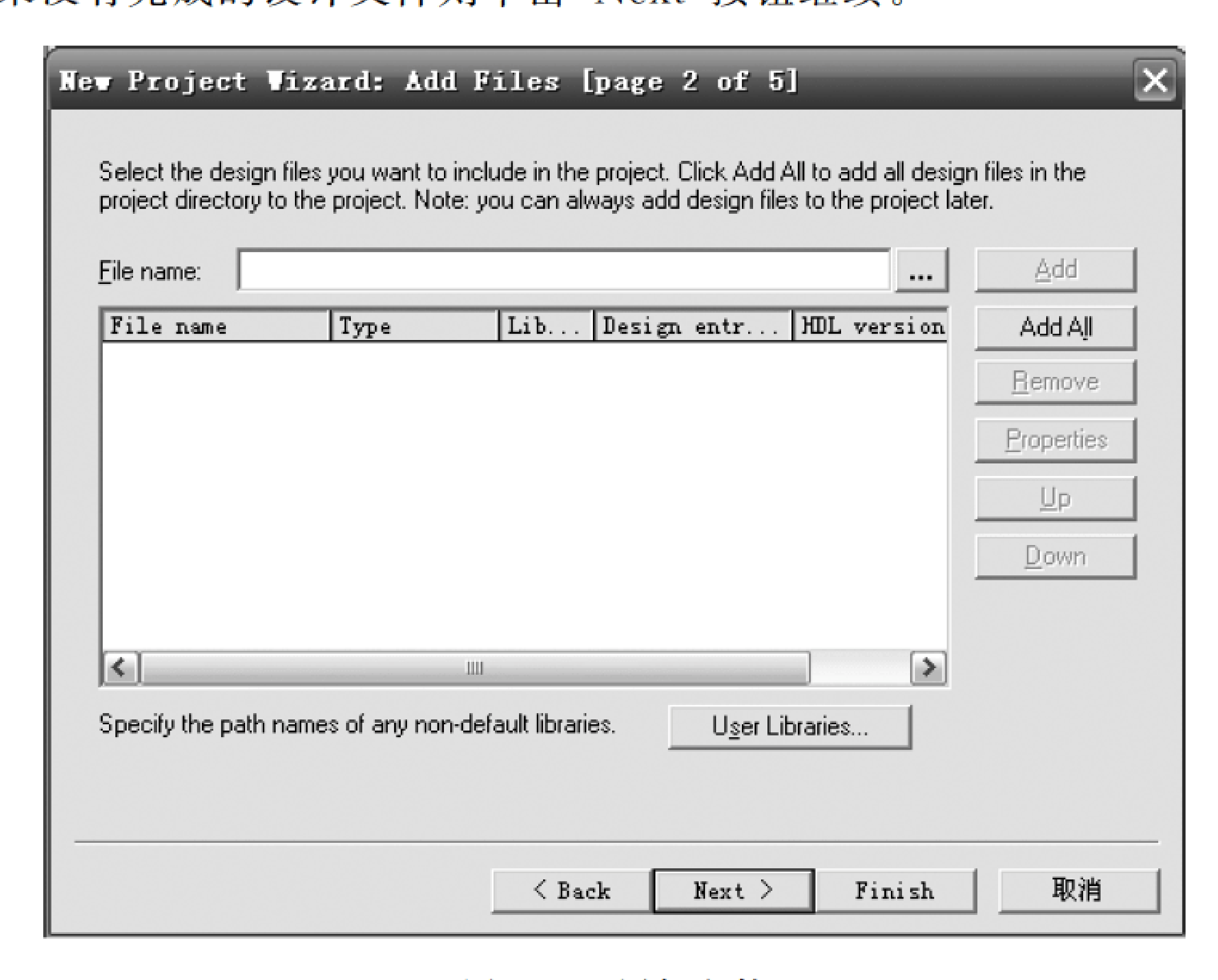


图 2-13　添加文件

选择目标芯片。本书中的实验平台选用的是 Altera 公司的 DE2-70 开发板，此开发板上使用的是 Cyclone Ⅱ 系列 EP2C70F896C6 芯片，所以这里选择 Cyclone Ⅱ EP2C70F896C6，如图 2-14 所示。

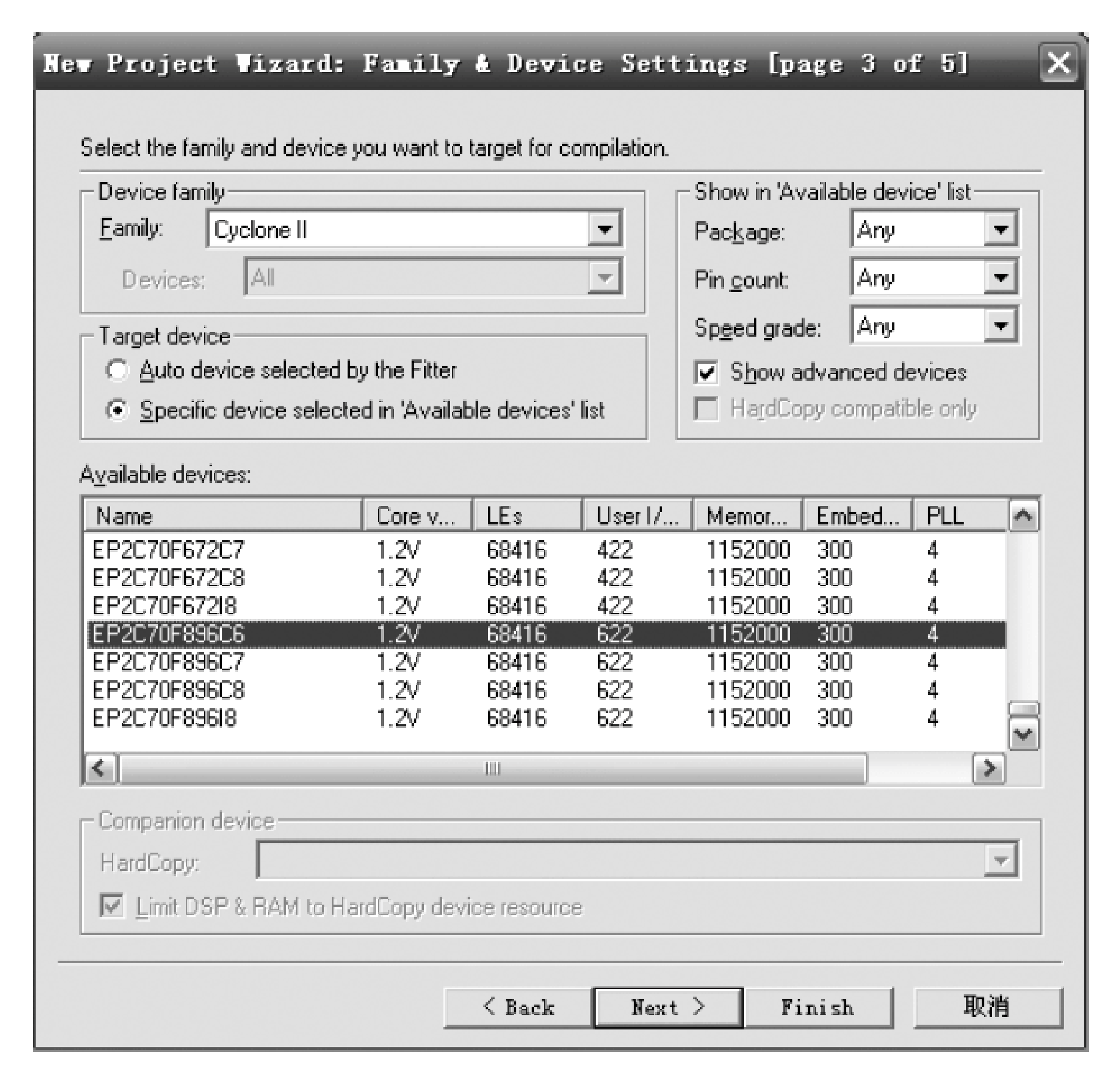


图 2-14　选择目标芯片

选择 EDA 工具。在这里可以由用户指定除 Quartus Ⅱ之外的用于设计输入、仿真、时序分析等的第三方 EDA 工具。这里默认采用 ModelSim-Altera 进行仿真，语言选择 Verilog HDL，单击“Next”按钮继续，如图 2-15 所示。

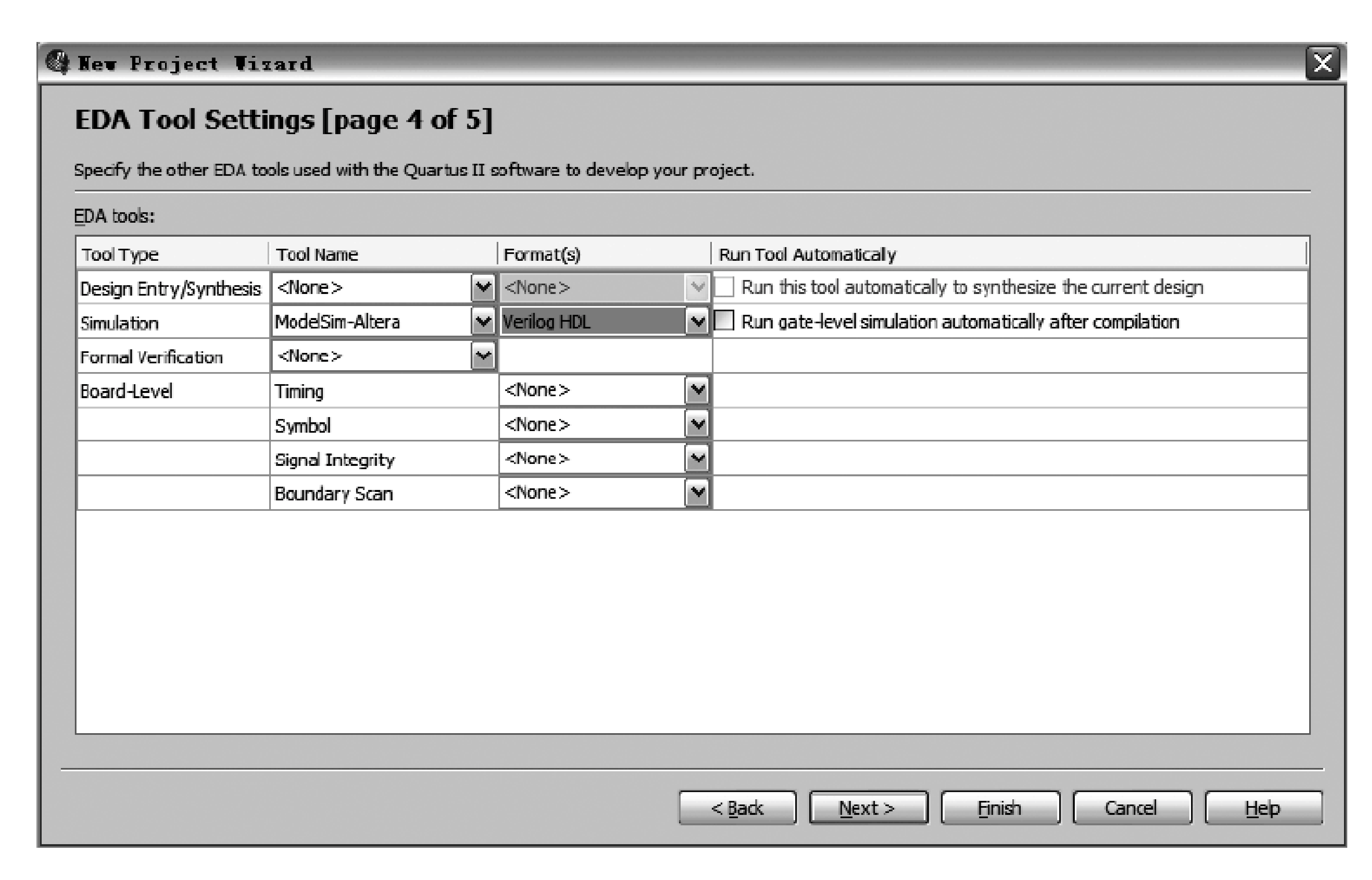


图 2-15　选择仿真模式

新建工程总结。新建工程完成后，Quartus Ⅱ会自动总结用户的设置，请确认设置是

否正确，特别是芯片的选择是否正确，如图 2-16 所示。

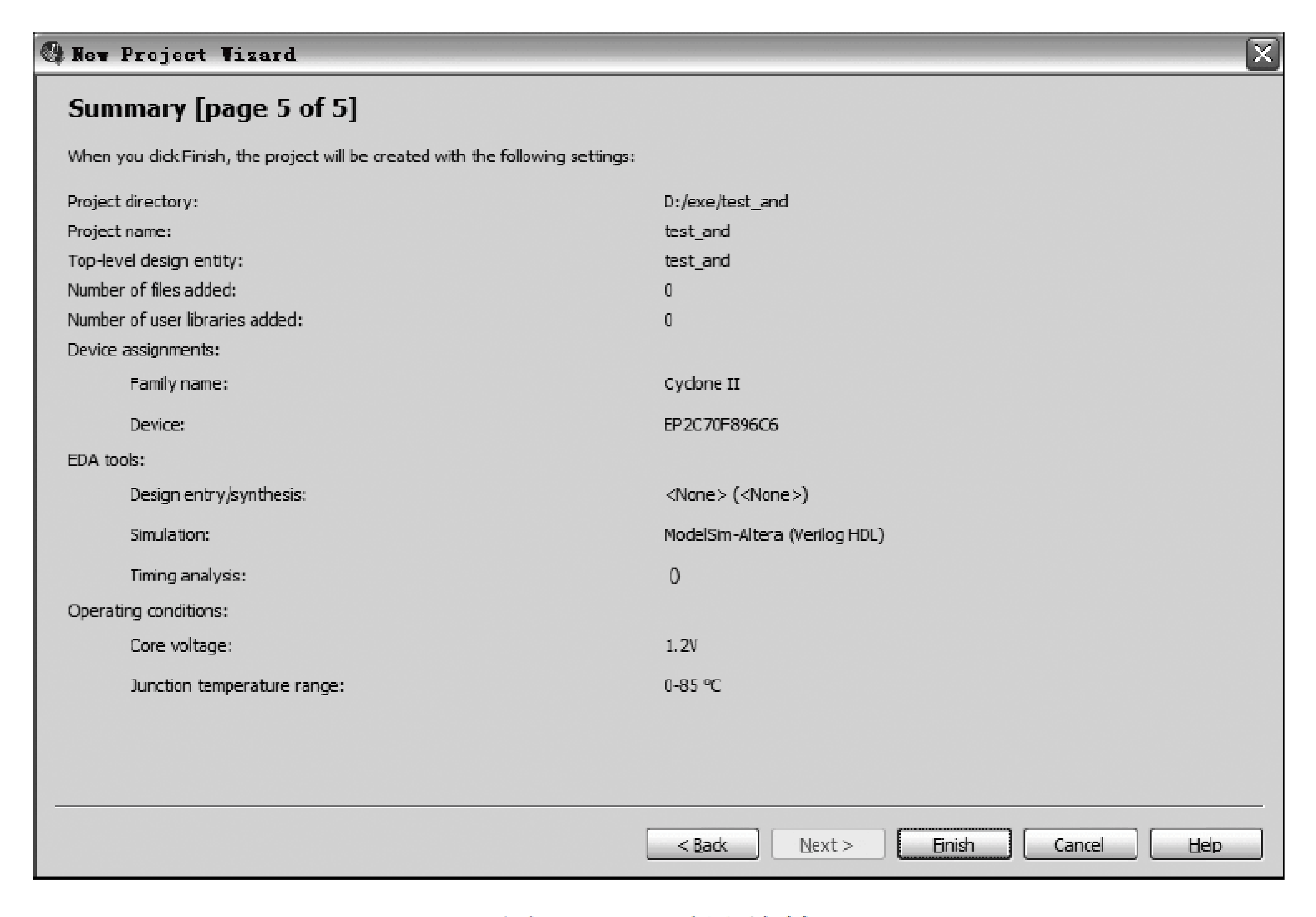


图 2-16　项目总结

工程新建完成后，Quartus Ⅱ界面中“Project Navigator”的“Hierarchy”选项卡中会出现当前工程的工程名以及所选用的器件型号，请再次确认器件型号，如图 2-17 所示。

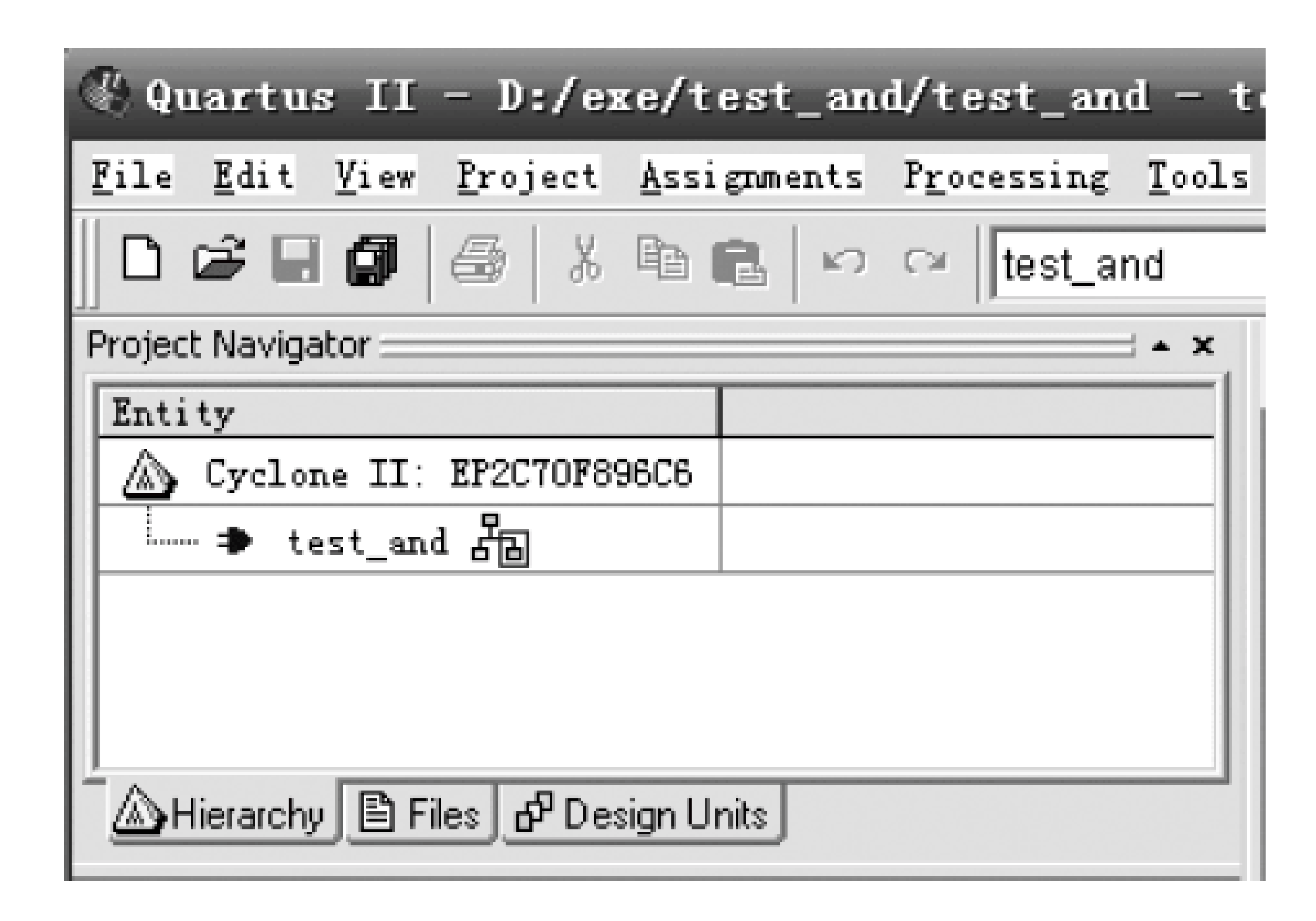


图 2-17　项目导航

培养良好的文件布局习惯。Quartus Ⅱ默认把所有的编译结果文件统一放在工程根目录下，为了让 Quartus Ⅱ像 Visual Studio 等 IDE 一样有一个专门的路径存放编译结果文件，需要在工程根目录下新建一个目录，并指定其为编译结果输出路径。

单击菜单项“Assignments→Settings”，选中“Compilation Process Settings”选项卡，勾选“Save Project output files in specified directory”选项，输入路径（一般为“debug(.\debug)”或者“release(.\release)”），如图 2-18 所示，单击“OK”按钮完成设置。

2.2.2.2　设计输入

工程新建结束后，新建工程输入文件，输入用户设计的原理图或者硬件描述语言文件。Quartus Ⅱ软件支持多种设计输入模型，本次实验使用 Quartus Ⅱ软件支持的 Block Diagram/Schematic File 编辑器（框图或者原理图编辑器）。

单击菜单项“File→New”、单击图标 🗋 或者使用快捷键“Ctrl＋N”新建一个设计文件，选择“Block Diagram/Schematic File”选项，如图 2-19 所示，单击“OK”按钮新建一个原理图/框图输入设计文件。

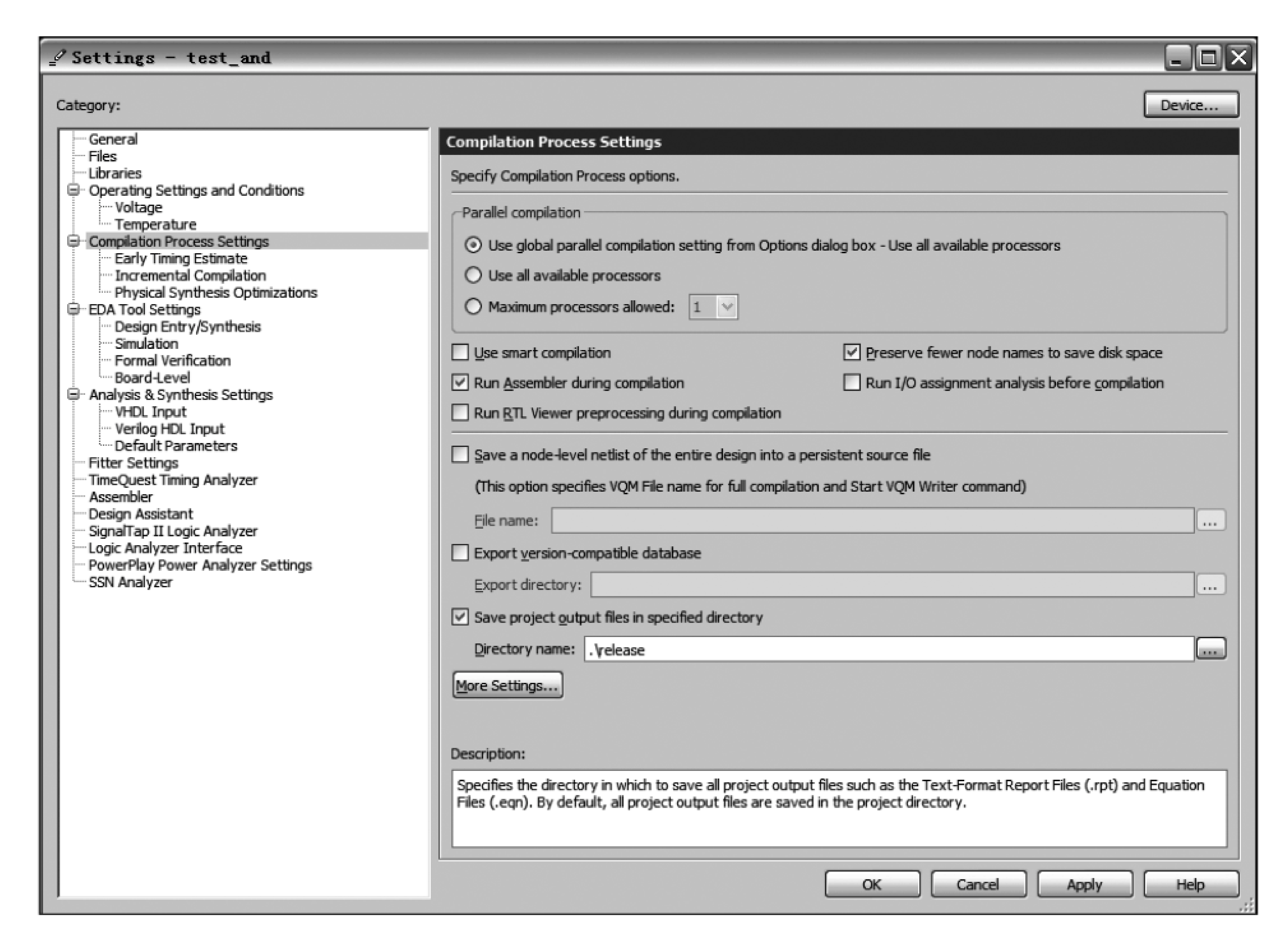


图 2-18　指定编译结果输出路径

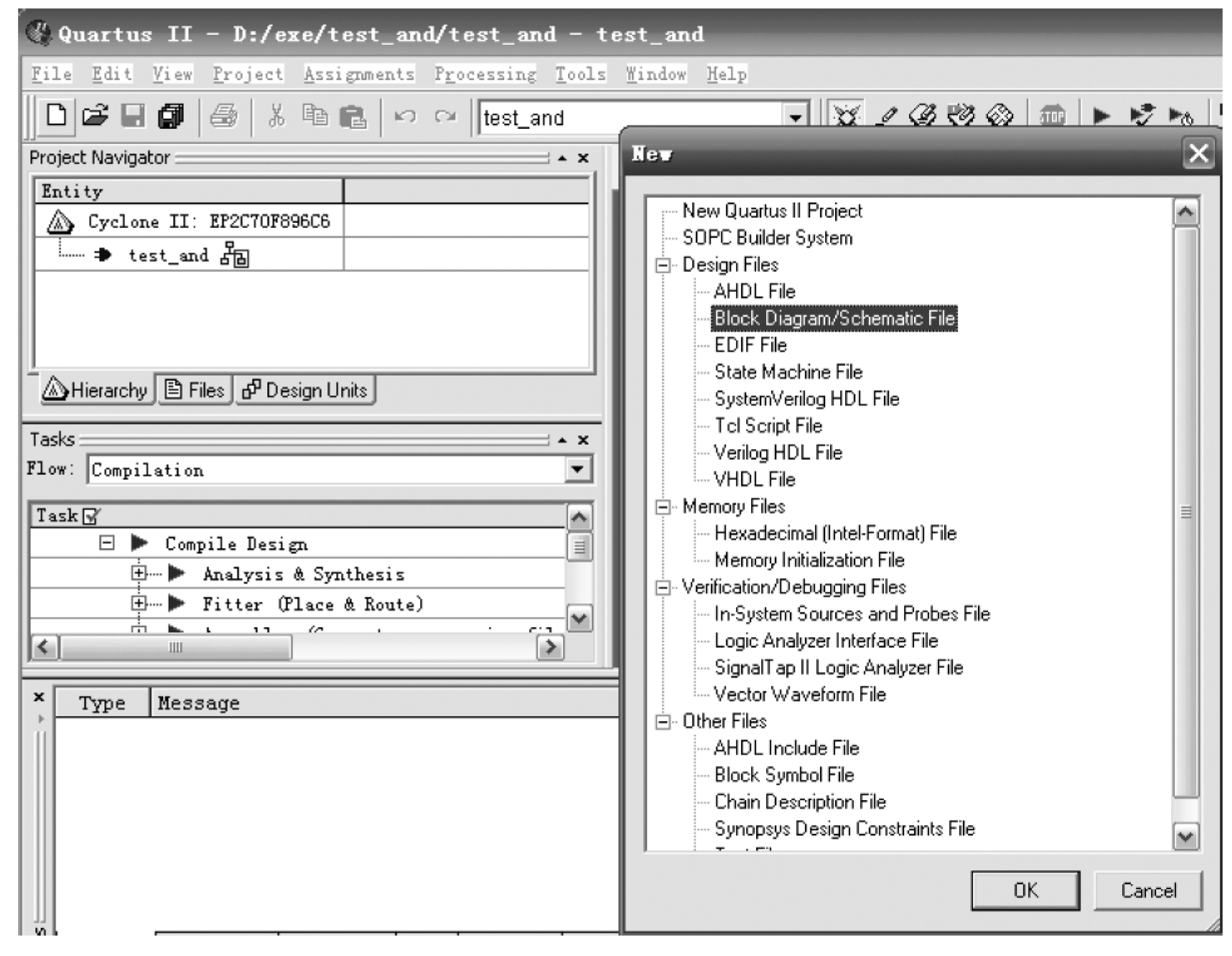


图 2-19　新建原理图文件

本实验设计的是一个三端控制灯的控制电路，其电路原理图如图 2-20 所示。

注：为了与 EDA 软件中电路元器件图形符号一致，本书中电路原理图中的元器件采用 ANSI/IEEE 91-84 标准图形符号。

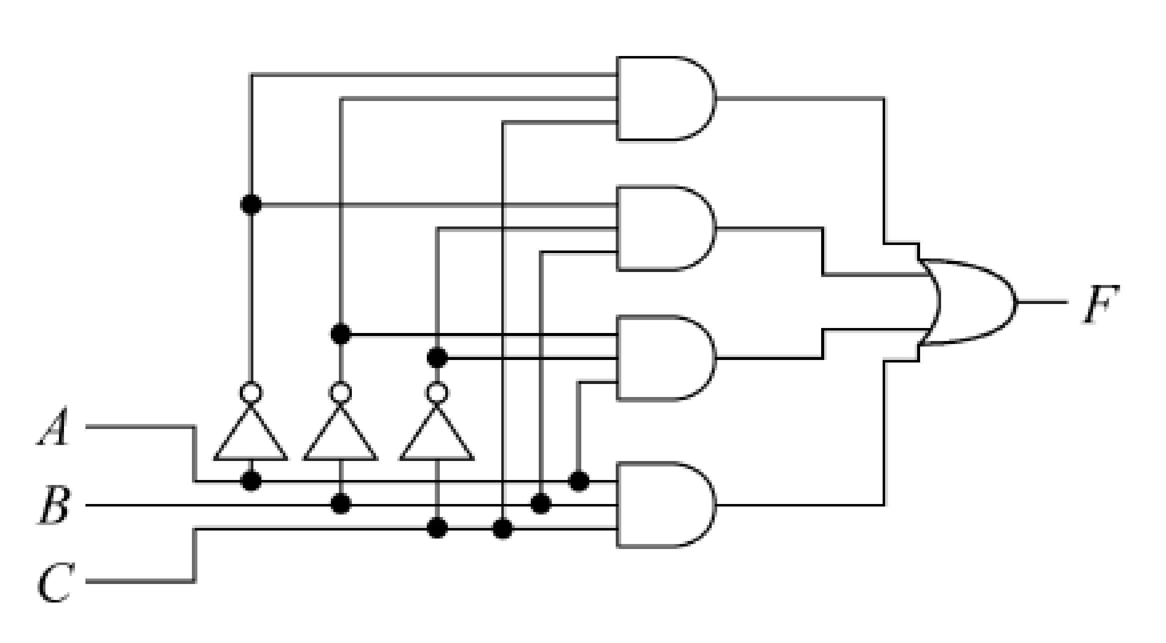


图 2-20　三路开关控制灯的控制电路原理图

用鼠标双击图形编辑器窗口的空白处或者单击图形编辑器窗口上侧工具条中的图标，选择相应电路符号进行设计，如图 2-21 所示。

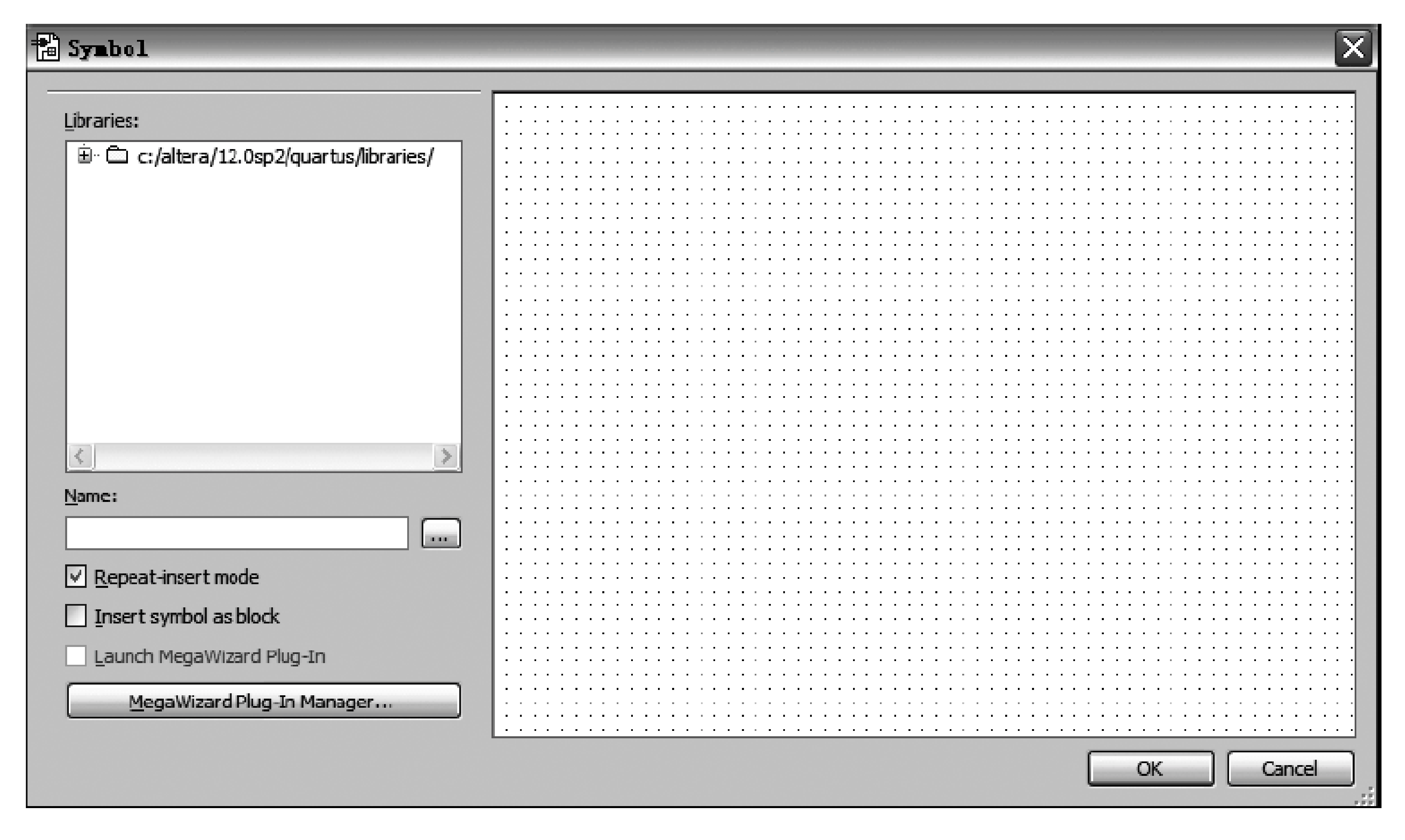


图 2-21　原理图编辑框

展开左侧树状目录到“…→primitives→logic→and3”，找到三输入与门，单击“OK”按钮后，在图形编辑窗口单击放置即可，如图 2-22 和图 2-23 所示。

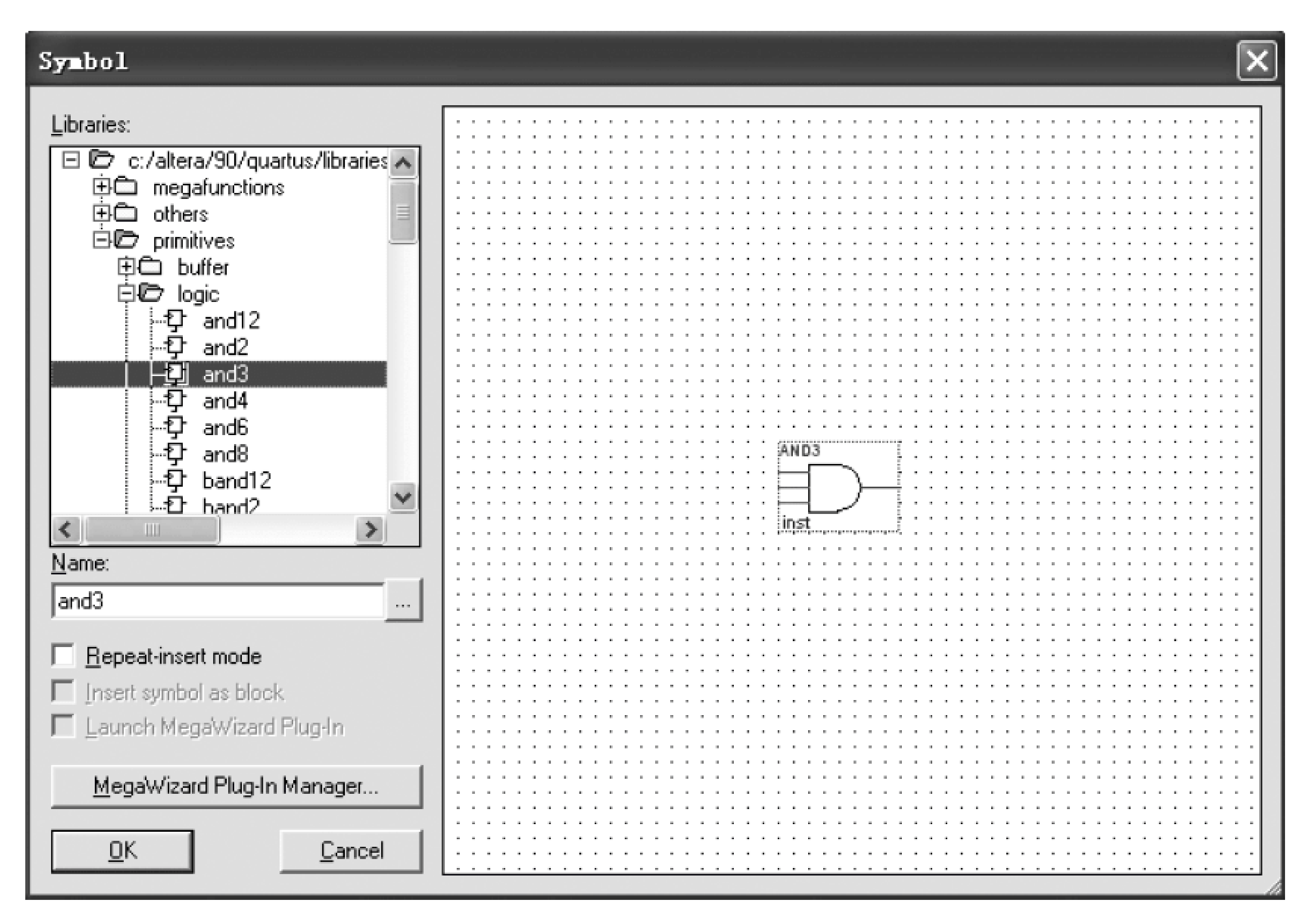


图 2-22　选择与门

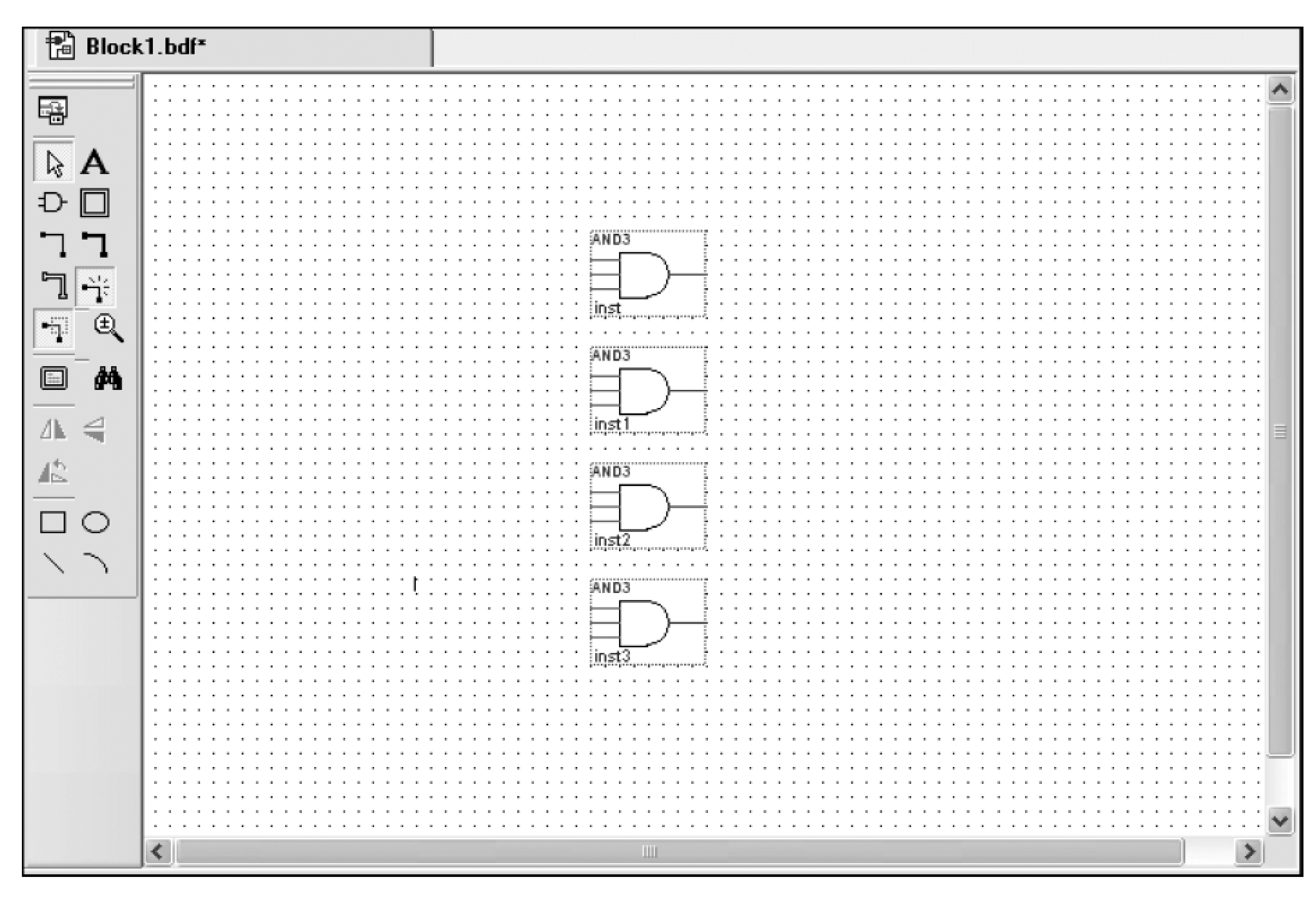


图 2-23　添加与门

展开左侧树状目录到“…→primitives→logic→or4”，添加四输入或门，如图 2-24 所示。

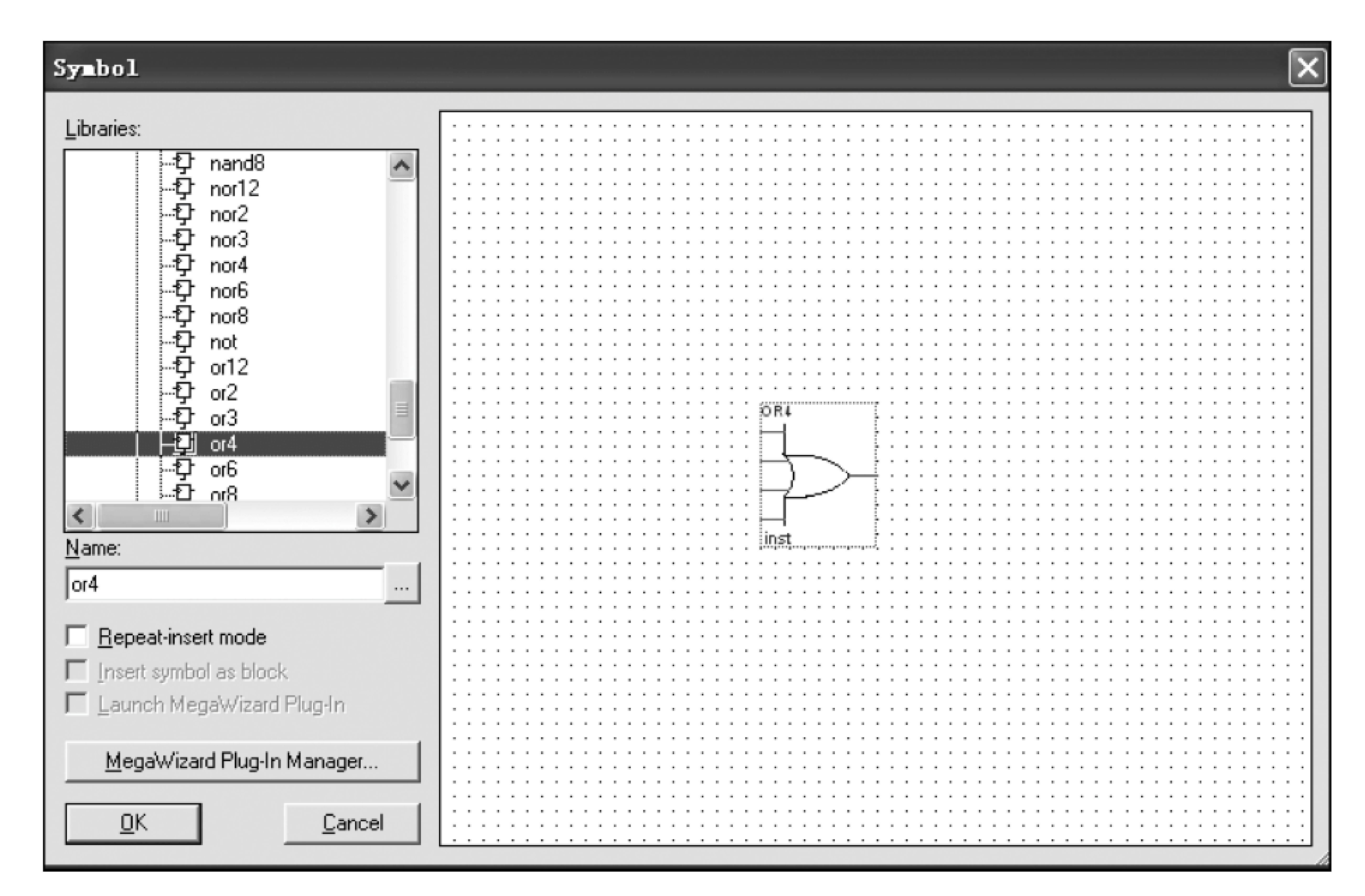


图 2-24　选择或门

按照上述步骤再添加一个非门并且旋转非门的方向，以方便画图，如图 2-25 所示。

放置输入输出（I/O）引脚，位置在目录“…→primitives→pin→input”与“…→

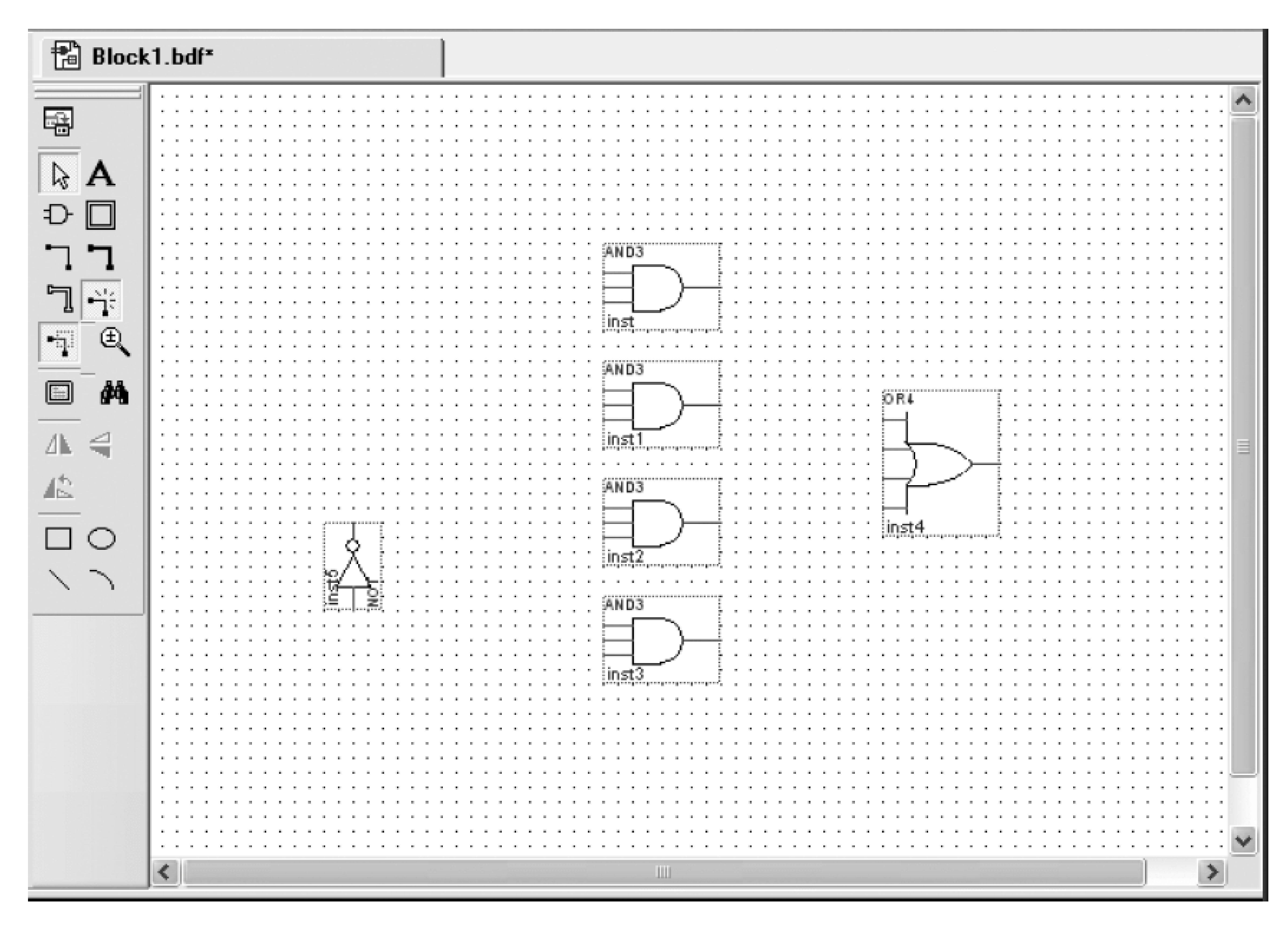


图 2-25 添加非门

primitives→pin→output”下，如图 2-26 与图 2-27 所示。

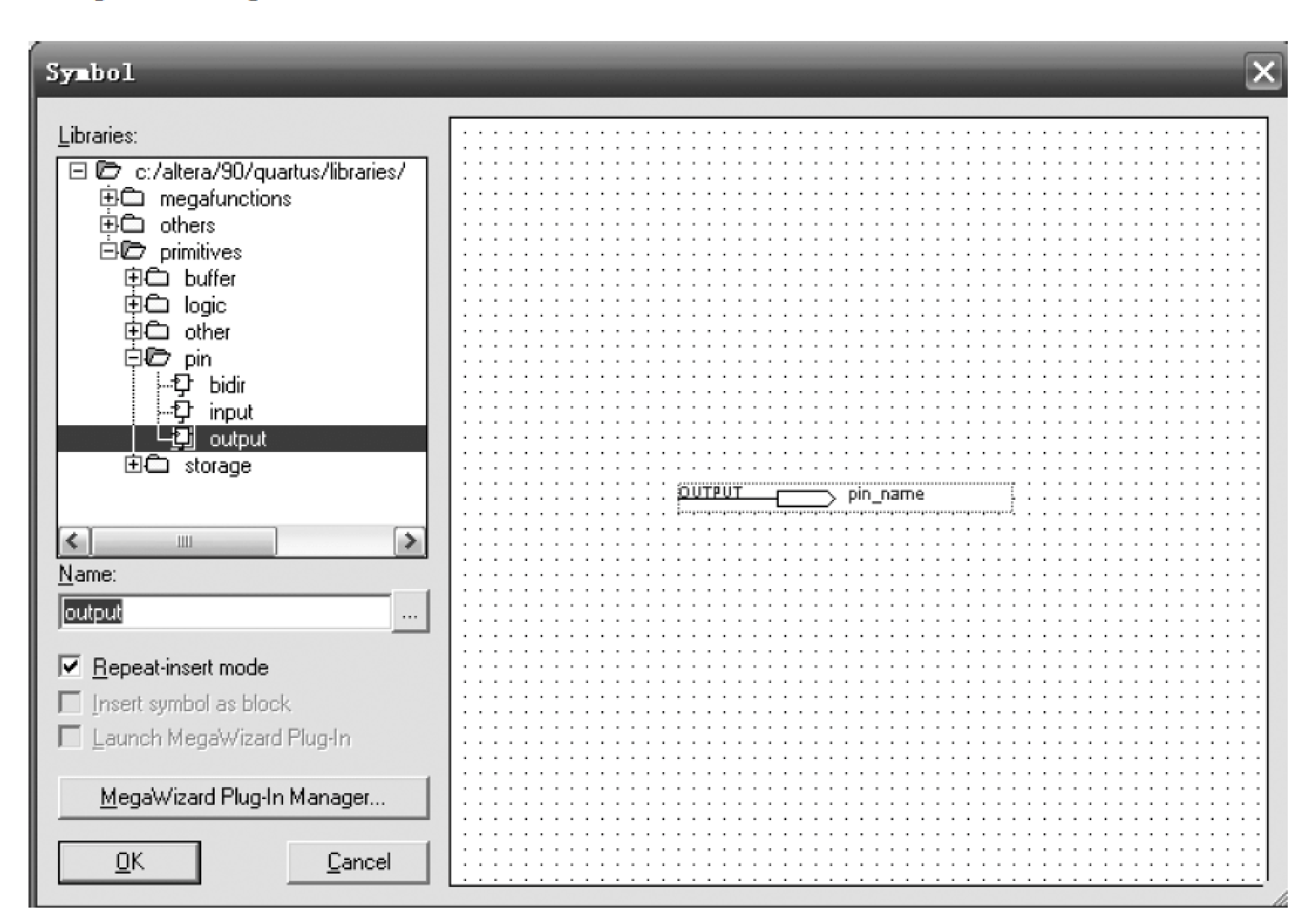


图 2-26 选择 I/O 引脚

进行电路图中元件之间的连接，用鼠标拖动电路符号的引线或者利用⌉工具即可连接元件，在元件连接完成后请仔细检查是否所有电路符号都已正确相连，没有连接的地方在图中会显示出叉(×)，请确定你设计的电路中没有叉(×)，完成后完整的电路图如图 2-28 所示。

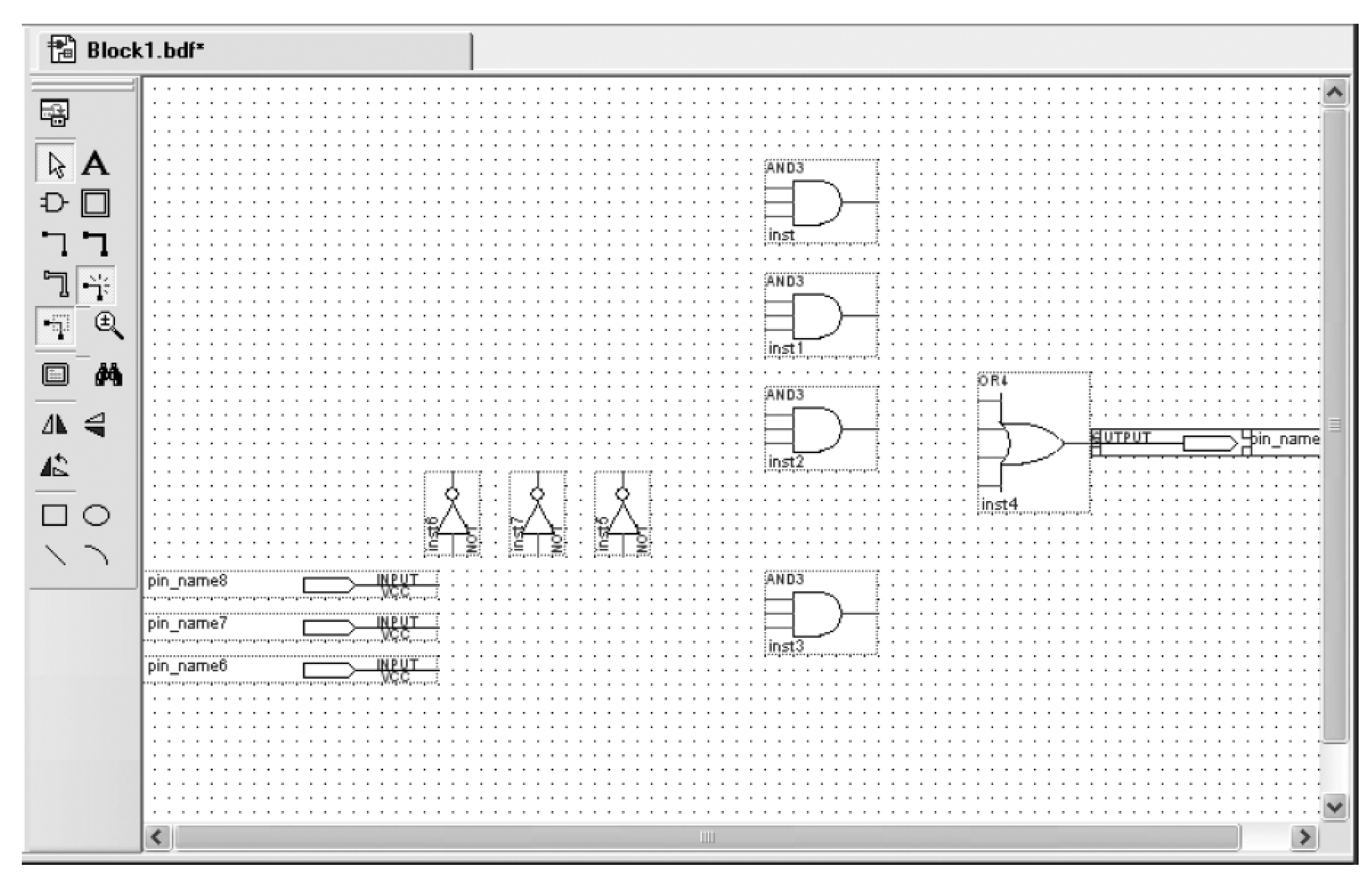


图 2-27　添加引脚

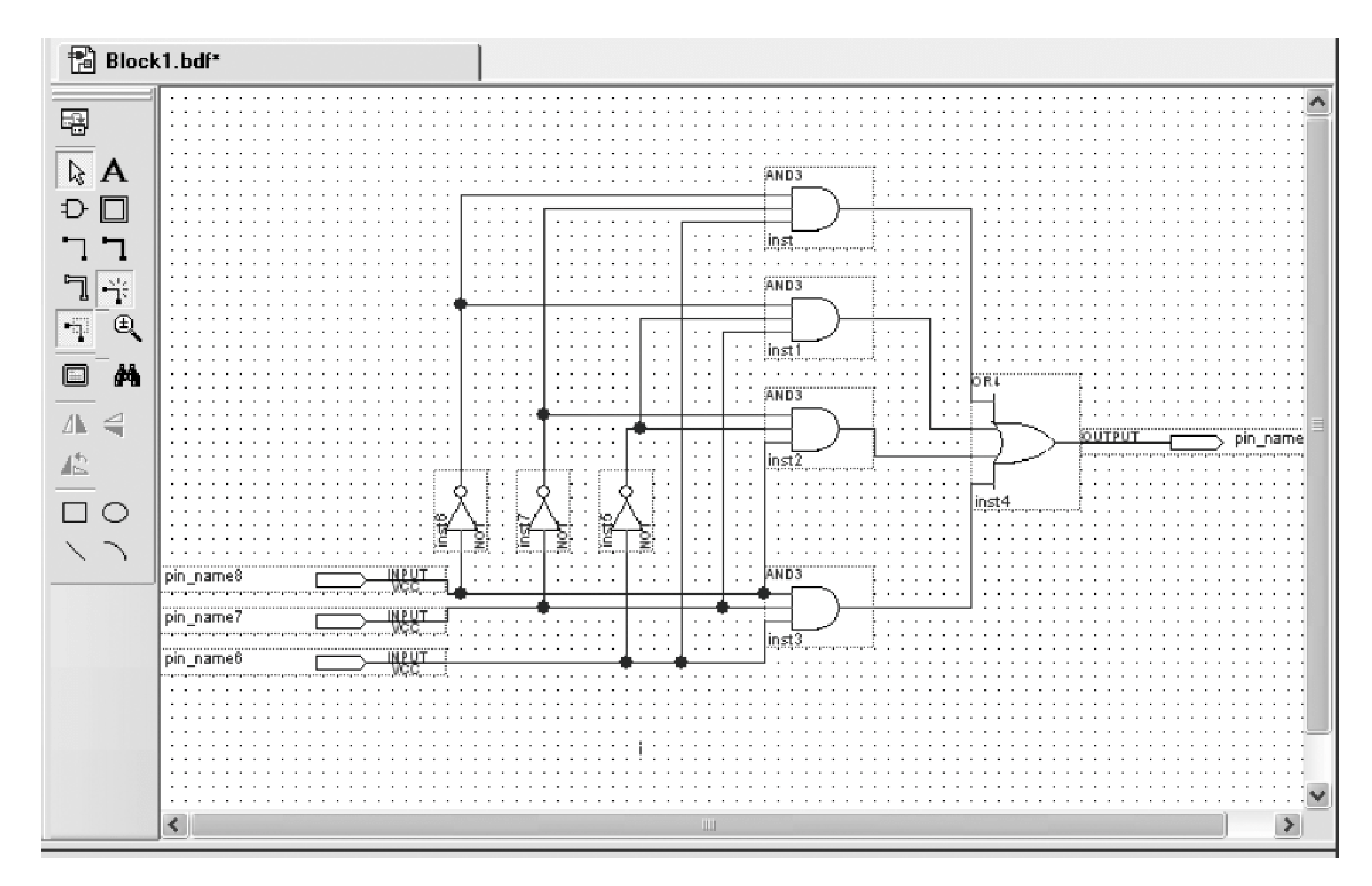


图 2-28　完整电路图

修改电路节点的名称，双击元件即可修改电路符号和节点的名称。这里，我们根据开发平台提供的引脚配置文件“DE2_70_pin_assignments.csv”将输出节点名称改为 oLEDG[0]，双击器件，在跳出的对话框中即可修改器件名称。将三个输入节点名称分别改为 iSW[2]、iSW[1]和 iSW[0]，并且保存设计文件为“顶层实体名.bdf”，这里的顶层实体名就是在建立工程时所取的顶层实体名，本例中为“test_and”，如图 2-29 所示。

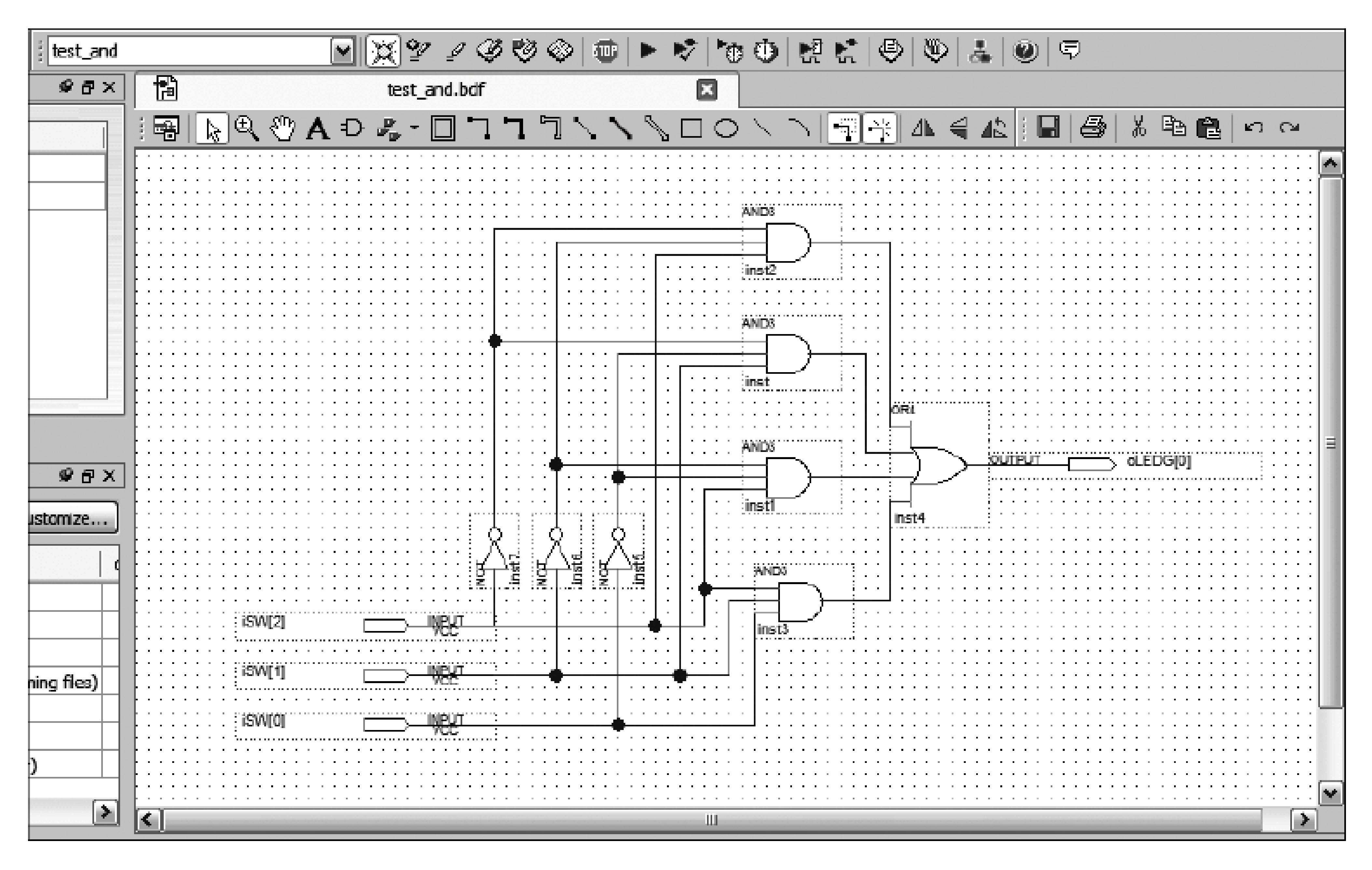


图 2-29　修改引脚名称并保存

2.2.2.3　分析与综合

单击图标、单击菜单项“Processing→start→Start Analysis & Synthesis”或者使用快捷键 Ctrl+K 执行分析与综合，如图 2-30 所示。

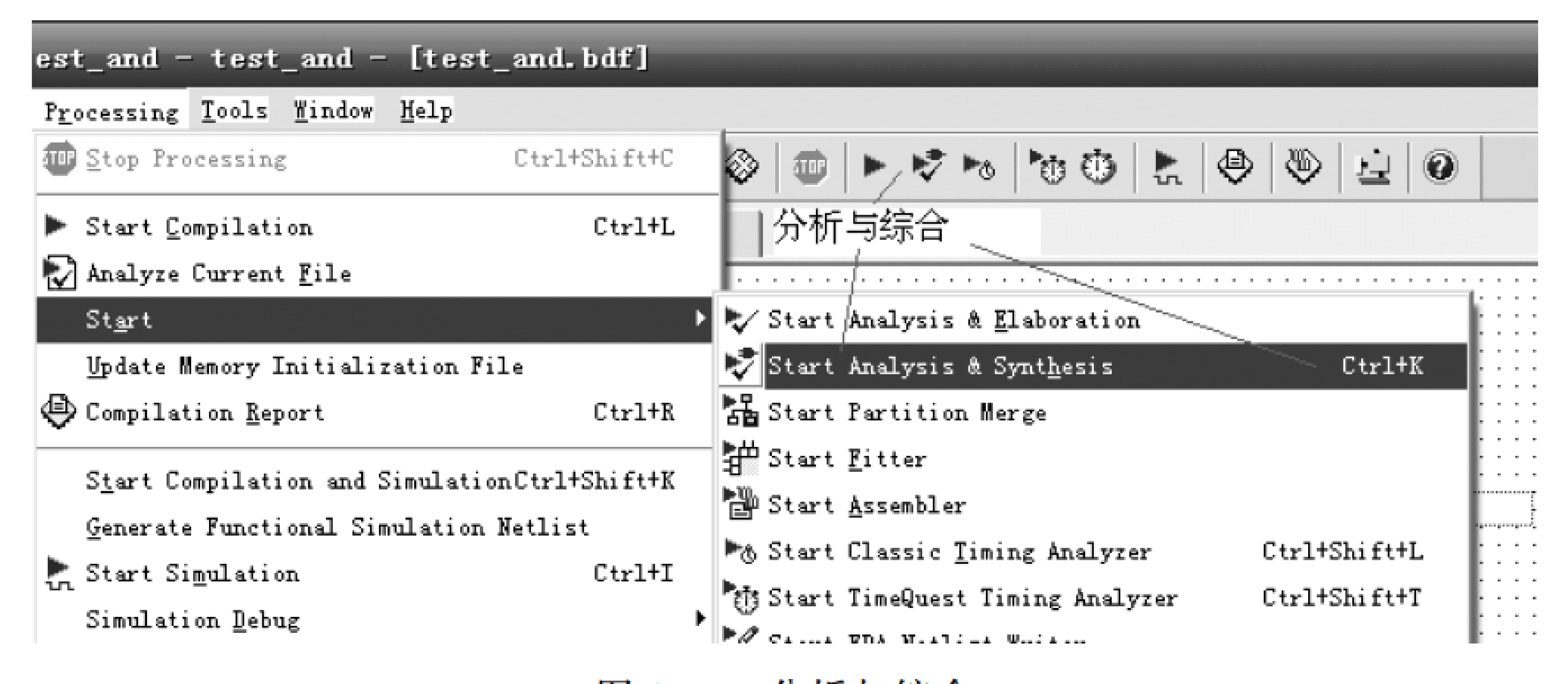


图 2-30　分析与综合

“Analysis & Synthesis”的 Analysis 阶段将检查工程的逻辑完整性和一致性，以及边界连接和语法错误。“Analysis & Synthesis”还对设计实体或工程文件的逻辑进行综合和技术映射。它从 Verilog HDL 和 VHDL 中推断触发器、锁存器和状态机。

分析与综合的时候，Quartus Ⅱ会给出编译进度，在“Compilation Report”窗口给出编译结果，显示编译进度，如图 2-31 所示。

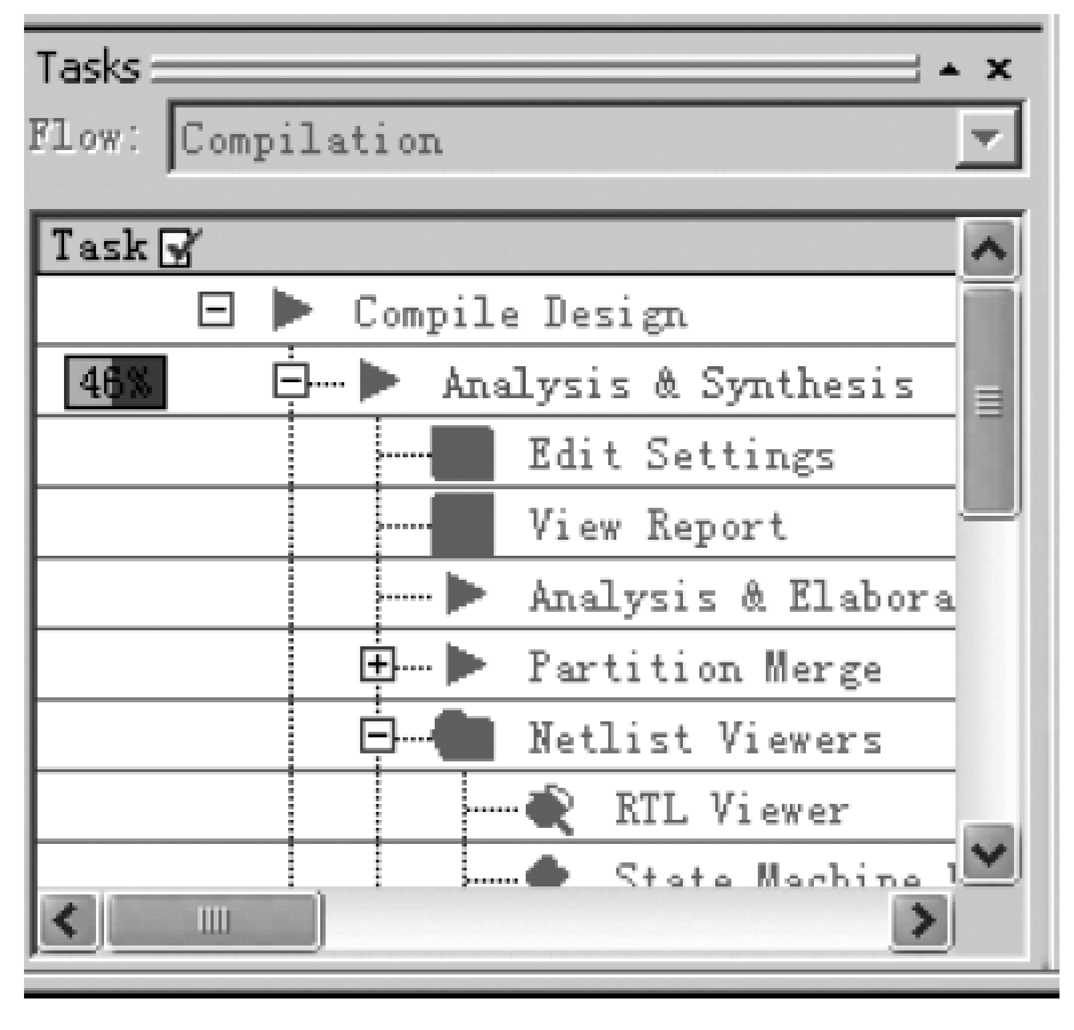


图 2-31　编译进度

“Message”窗口会给出编译过程的具体情况，包括“information，warning，error”等信息，如图 2-32

所示。

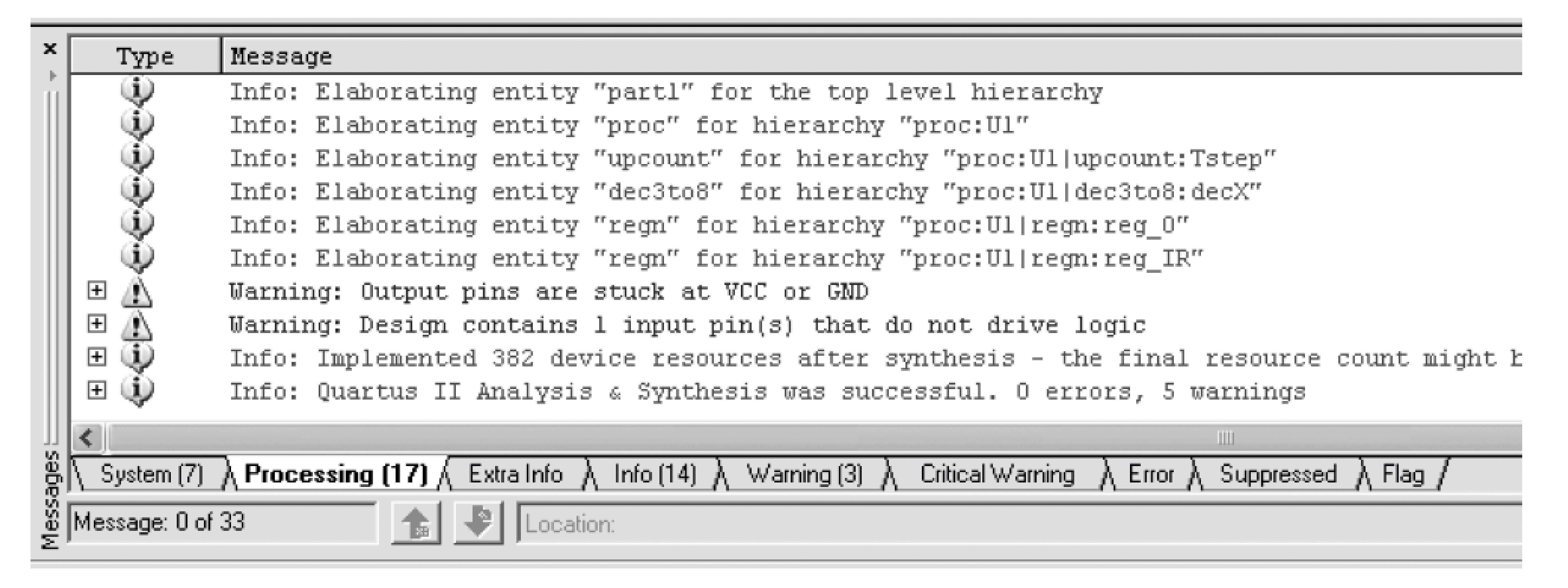


图 2-32　编译报告窗口

“information”一般描述的是编译的进展；“warning”是编译过程中的一些警告信息，有些信息对结果不产生影响，有些警告信息会影响设计的结果，请大家每次编译后都要仔细查看警告信息，排除一些“重要”的警告，这样才能产生正确的结果；“error”一般指出设计或编译中发现的语法错误，说明此设计或者工程中的某些参数配置是不正确的，必须改正这些错误，重新编译。

分析与综合完成后，Quartus Ⅱ给出综合报告，详细显示使用的 LE 个数和引脚等信息，如图 2-33 所示。

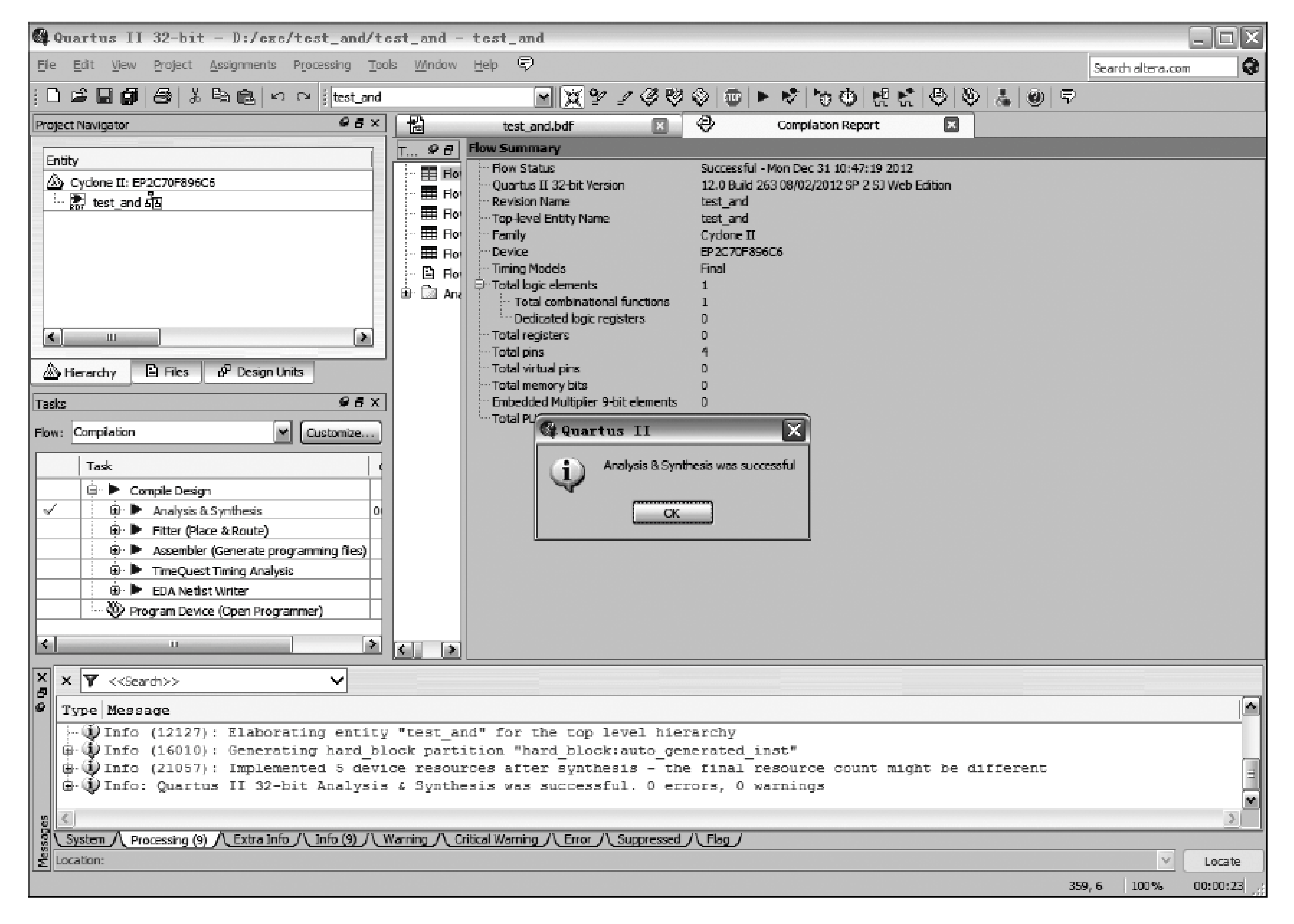


图 2-33　综合完成

2.2.2.4 功能仿真

为了测试工程在功能上是否满足设计需要，在分析与综合完成后要对其进行功能仿真，也称为前仿真。Quartus Ⅱ 12.0 默认采用 Modelsim－Altera 10.0d 进行仿真，Modelsim 是 Mentor 公司设计的业界最优秀的 HDL 语言仿真软件，有友好的仿真界面。但是，Modelsim 不提供对 Quartus Ⅱ产生的电路原理图文件（Block Diagram/Schematic File）进行仿真，只对用 HDL 语言文件产生的工程进行仿真。本例中采用的设计文件是“Block Diagram/Schematic File”，必须先产生此文件的 Verilog 语言文件，才能利用 Modelsim 对工程进行仿真。

产生此文件的 Verilog 语言文件，退出编译结果报告界面，进入原理图源文件，单击“File→Creat/Update→Creat HDL Design File from Current File”，如图 2-34 所示。

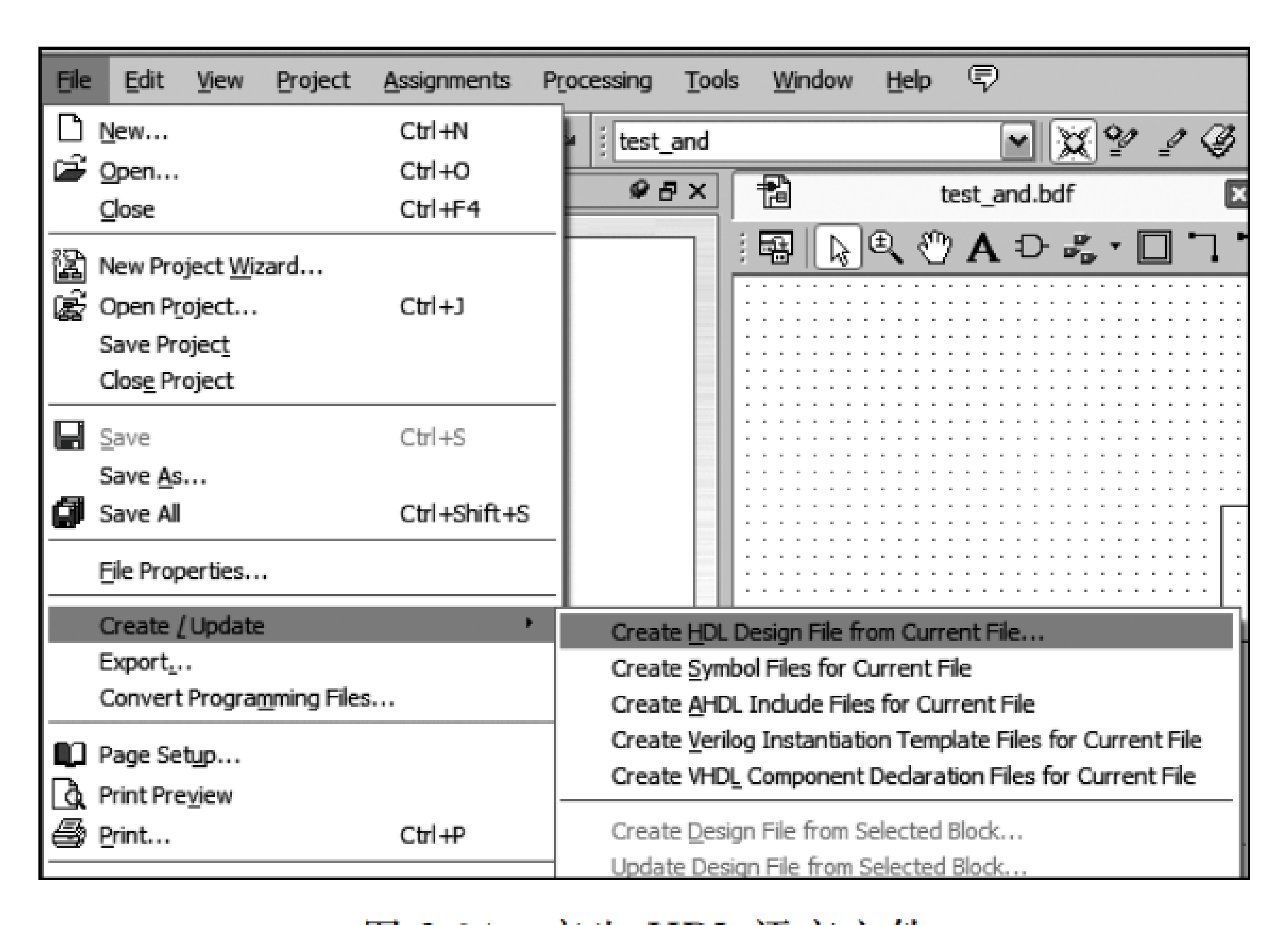


图 2-34 产生 HDL 语言文件

在弹出的对话框中选择 Verilog HDL，新产生的 Verilog HDL 文件名默认与原文件名相同，这里为“test_and.v”，如图 2-35 所示，单击“OK”按钮完成。完成后，读者可以通过“File→Open”找到“test_and.v”文件，将其打开，看看 Quartus Ⅱ根据电路原理图产生的 Verilog HDL 语言文件的是如何实现其电路功能的。

将产生的 Verilog HDL 文件加入到工程中，在 Project Navigator 的 File 栏中，右键单击菜单文件“File”选择“Add/Remove Files in Project”，如图 2-36 所示。

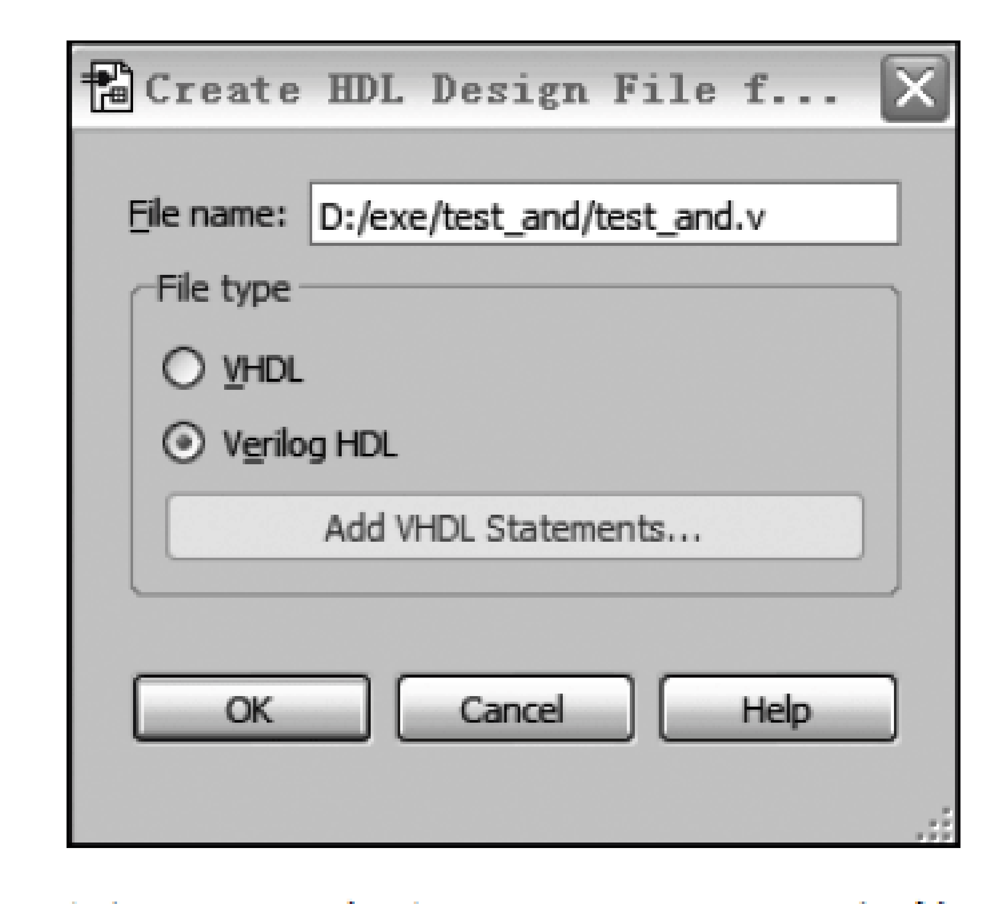


图 2-35 产生 Verilog HDL 文件

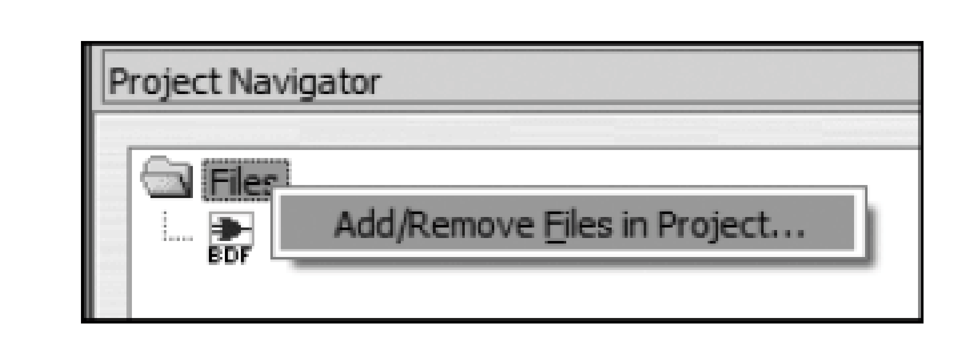


图 2-36 增加文件到工程中

单击进入增加文件到工程中对话框，单击“File name：”栏后的 ... 图标，进入选择要添加的文件位置对话框，找到刚才产生并保存在工程目录下的“test_and. v”文件，如图 2-37 所示。

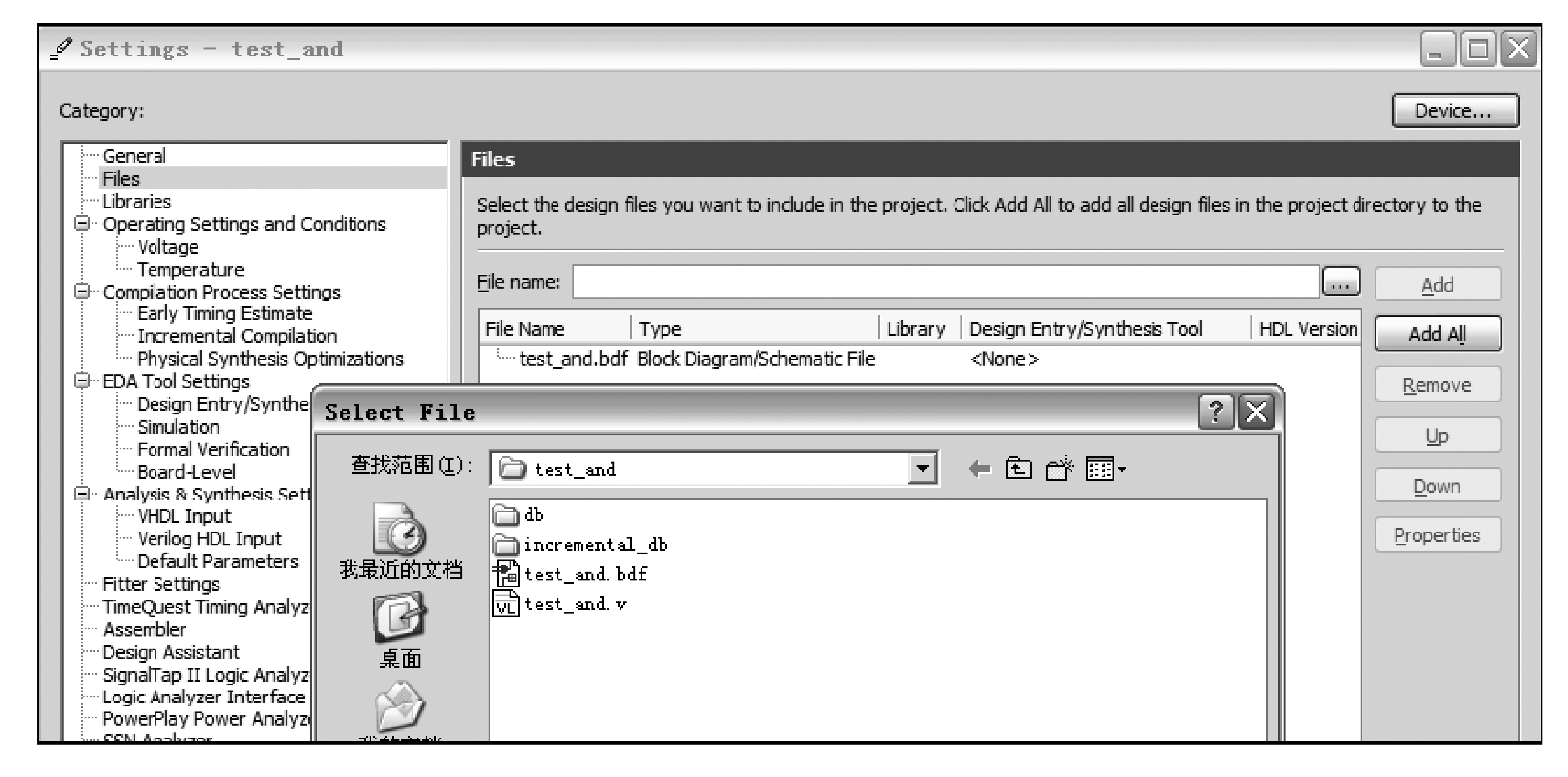


图 2-37　选择 Verilog HDL 文件

将此文件“打开”、“Add”到工程中，此时，工程中有两个称为“test_and”的实体，一个是“test_and. bdf”，另外一个是“test_and. v”。如果这时重新编译，编译器就会报错：“Error(12049)：Can't compile duplicate declarations of entity "test_and" into library "work"”。必须将原来的“test_and. bdf”移出工程，选择“test_and. bdf”文件，单击“Remove”按钮，如图 2-38 所示。工程就只有“test_and. v”一个顶层实体文件了，重新对工程进行分析与综合，成功。

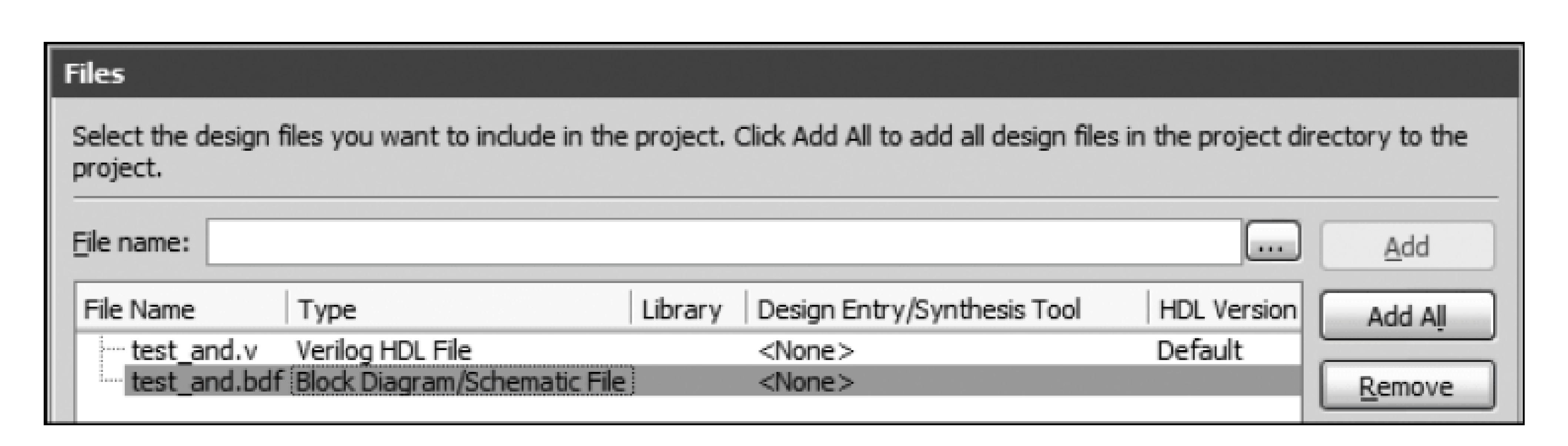


图 2-38　移出文件

产生用于仿真的测试代码，先由编译器自动产生用于测试的代码模板，单击“Processing”，选择“Start→Start Test Bench Template Writer”，如图 2-39 和图 2-40 所示。

测试代码 testbench 文件会默认自动生成在工程文件夹下的 simulation/modelsim 目录下，文件名默认为顶层实体名. vt，本例中为“test_and. vt”。

打开“test_and. vt”，根据生成的模板编写符合工程需要的测试代码，如图 2-41 所示。

设置仿真参数，单击“Assignment→Settings”，对仿真进行设置。选择时间尺度“Time scale”为 1μs，选择“Compile test bench”，如图 2-42 所示。

对 testbench 进行设置，单击“Test Benches”，出现如图 2-43 所示“Test Benches”对话框。

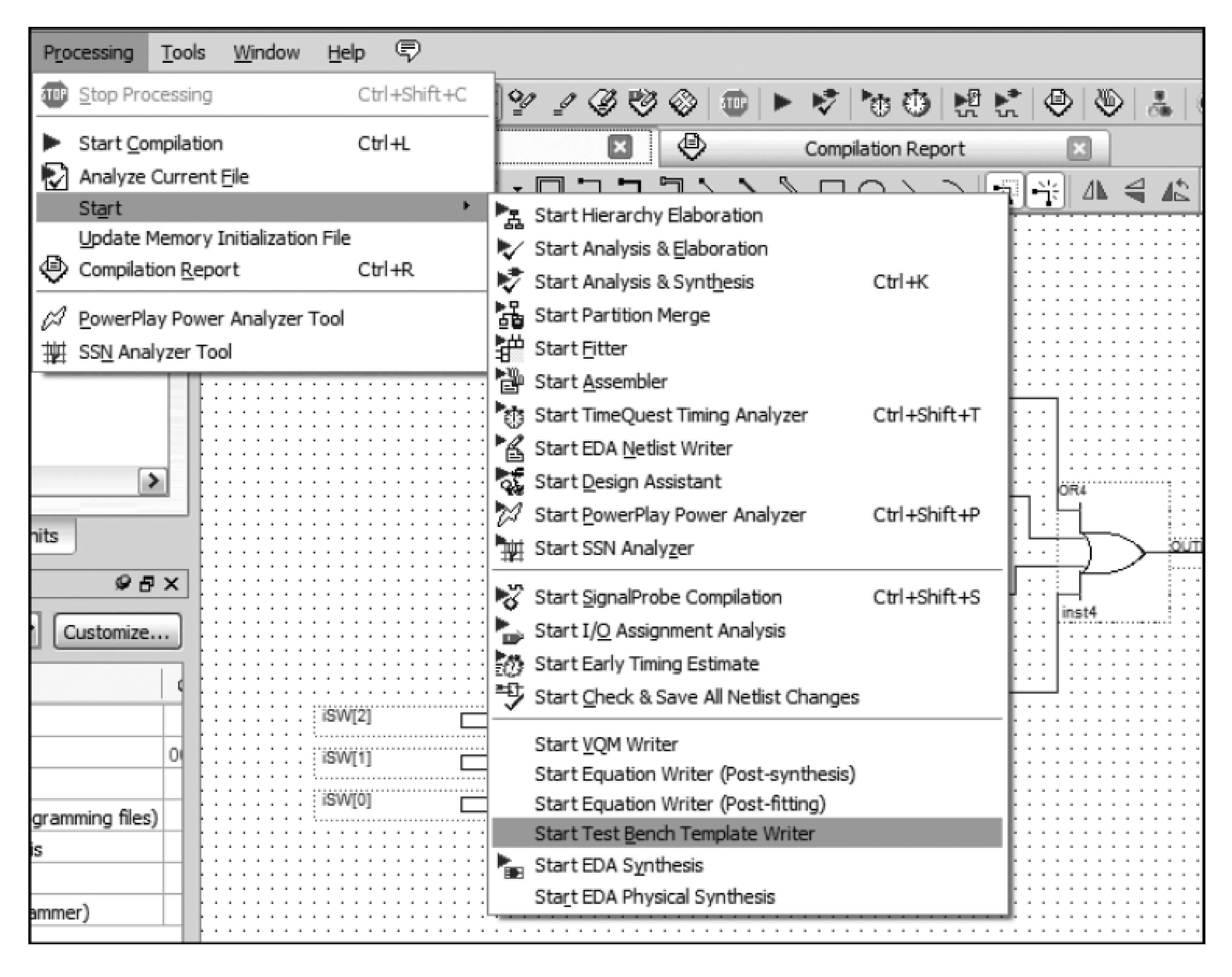


图 2-39　产生测试代码



图 2-40　测试代码模板编写成功

```

`timescale 10 ns/ 1 ps
module test_and_vlg_tst();
// constants
// general purpose registers
// test vector input registers
reg [2:0] iSW;
// wires
wire [0:0]  oLEDG;

// assign statements (if any)
test_and i1 (
// port map - connection between master ports and sig
    .iSW(iSW),
    .oLEDG(oLEDG)
);
initial

begin
// code executes for every event on sensitivity list
// insert code here --> begin

iSW[2]=0; iSW[1]=0; iSW[0]=0; #10;
iSW[2]=0; iSW[1]=0; iSW[0]=1; #10;
iSW[2]=0; iSW[1]=1; iSW[0]=0; #10;
iSW[2]=0; iSW[1]=1; iSW[0]=1; #10;
iSW[2]=1; iSW[1]=0; iSW[0]=0; #10;
iSW[2]=1; iSW[1]=0; iSW[0]=1; #10;
iSW[2]=1; iSW[1]=1; iSW[0]=0; #10;
iSW[2]=1; iSW[1]=1; iSW[0]=1; #10;
// --> end
end
endmodule
```

图 2-41　编写测试代码

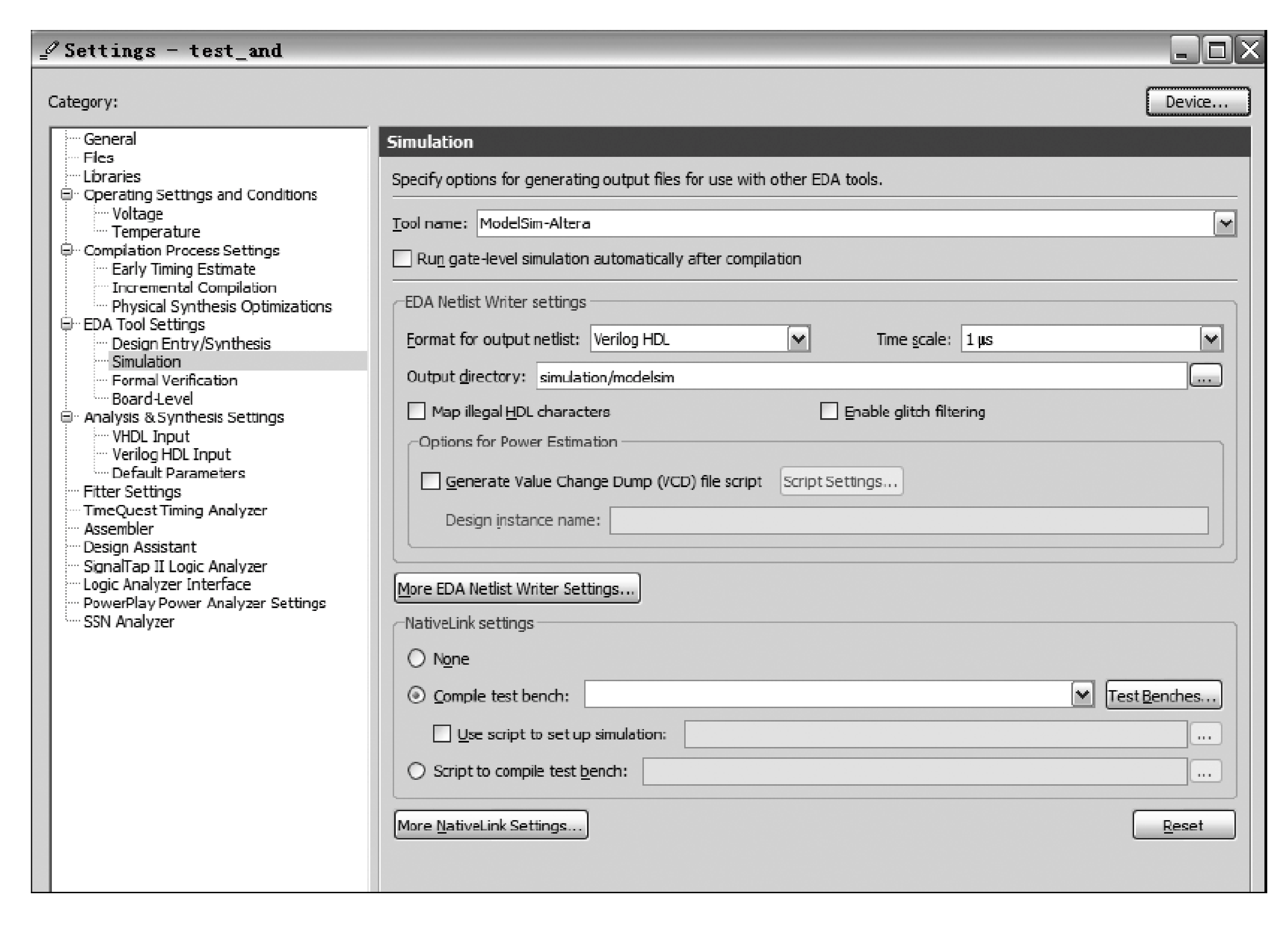


图 2-42　设置仿真参数

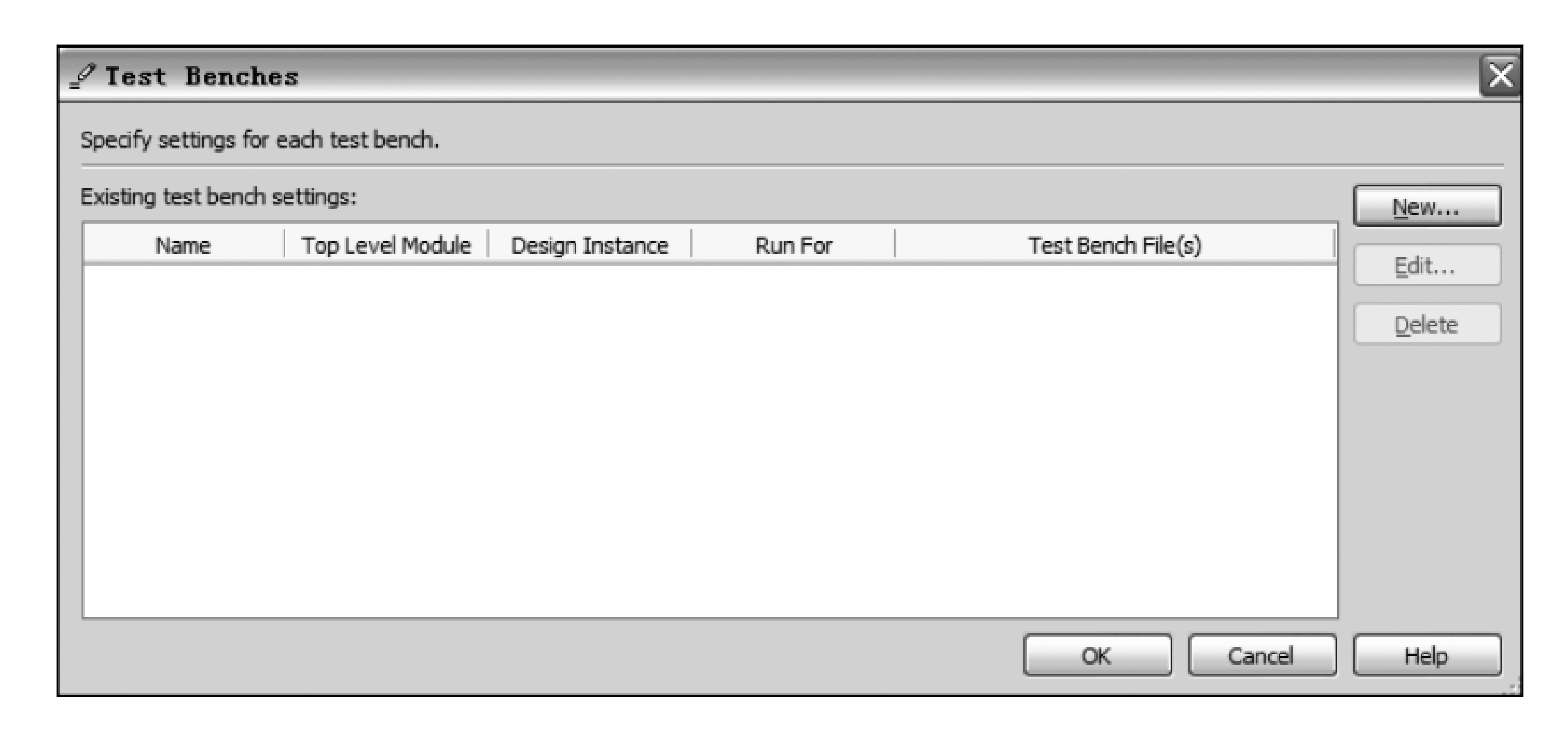


图 2-43　设置 testbench

单击“New”按钮，出现“New Test Bench Settings”对话框，在“Test bench name：”栏中填入刚才产生并修改的 testbench 的顶层实体名，即“test_and.vt”中的顶层实体名，这里是“test_and_vlg_tst”，此时“Top level module in test bench：”栏默认与“Test bench name：”栏相同。单击“File name：”栏后[...]图标，在出现的对话框中，选择已经按照工程修改好的 testbench 文件，打开，单击“Add”按钮，测试文件出现在了“Test bench and simulation files”栏中，如图 2-44 所示。

单击“OK”按钮，加入 testbench 文件，“Test Benches”对话框中出现了测试文件信息，如图 2-45 所示。单击“OK”按钮，测试文件设置完成。

回到工作 Quartus Ⅱ工作界面，单击，Modelsim 会自动运行，并且开始仿真，仿真结果如图 2-46 所示。

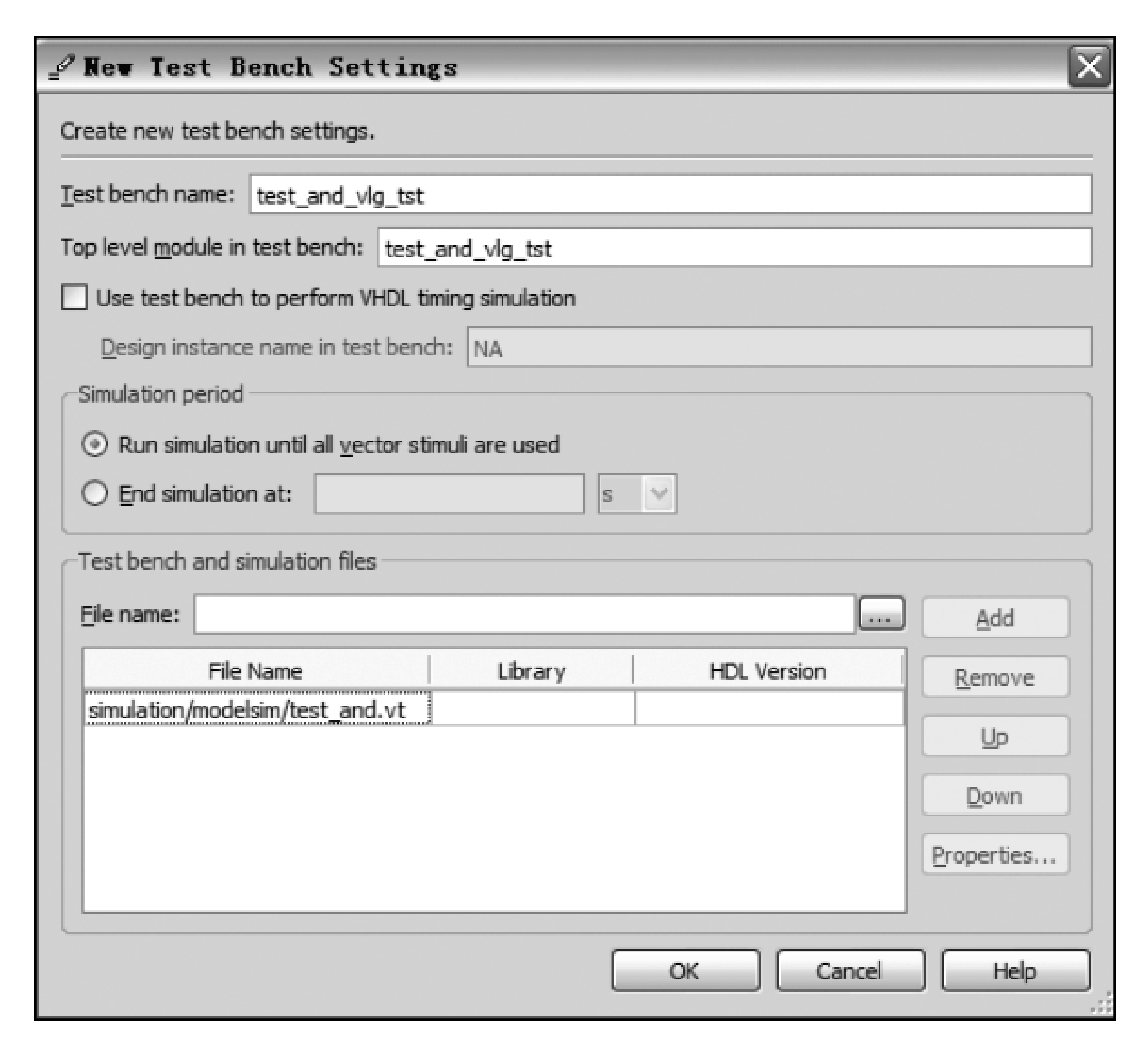


图 2-44　指定测试文件

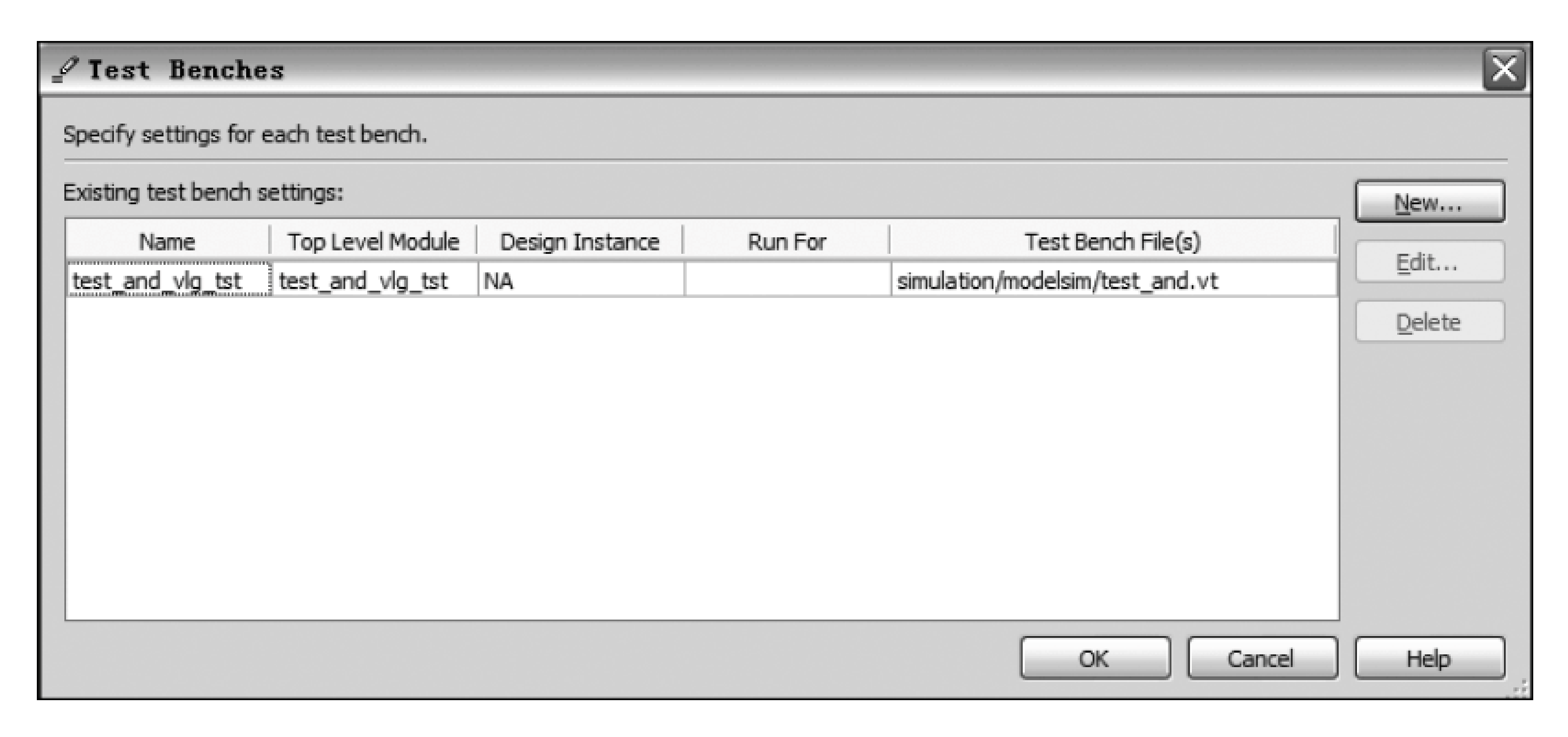


图 2-45　加入测试文件

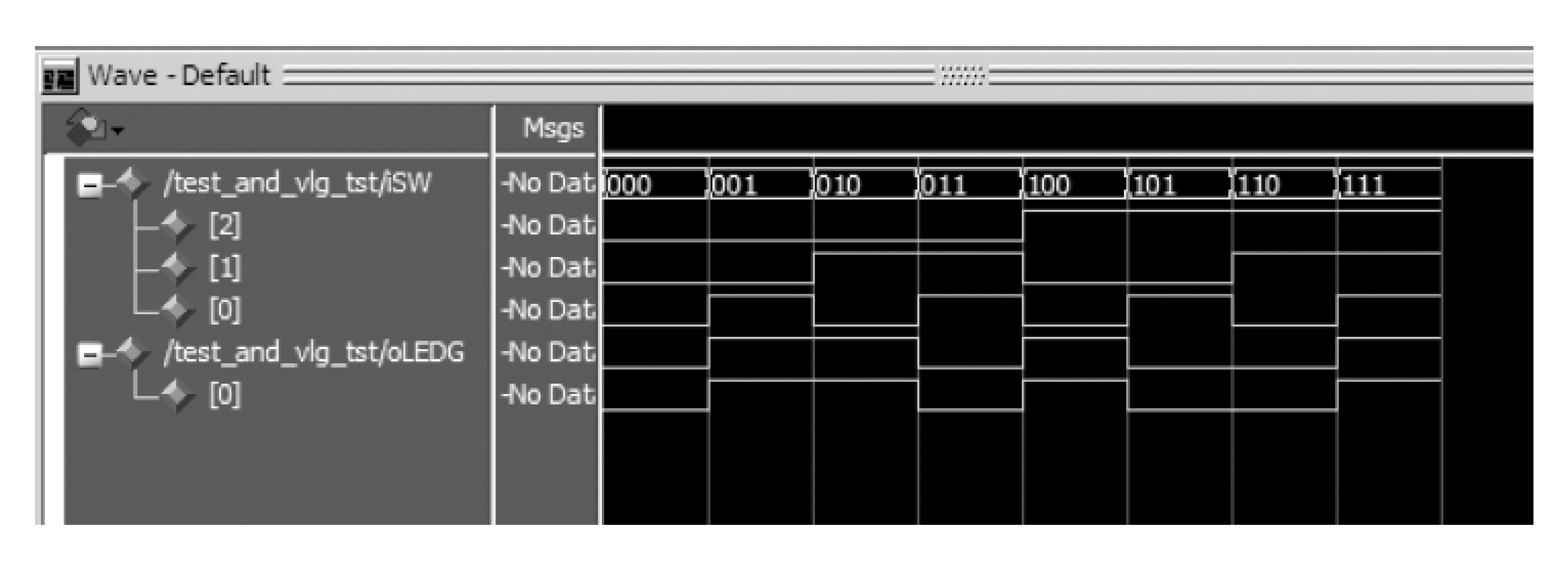


图 2-46　功能仿真结果

分析仿真结果，和真值表完全一致。在仿真图中我们还可以发现，在输入变化的同时输出信号也发生了变化，没有任何时间延迟。功能仿真是验证电路在逻辑功能上是否满足要求，不考虑电路延时所带来的影响。

2.2.2.5　分配引脚

引脚分配是 FPGA 开发过程中的一个重要步骤，设计者的设计文件是对 FPGA 的内部电路进行配置，如果需要利用 FPGA 开发平台上的外围电路对已经设计好的电路进行验证，就要将设计好的电路的输入与输出引脚连接到外围电路上。本实验中的输入引脚名为 iSW[2]、iSW[1]和 iSW[0]，即可利用开发板上的三个拨动开关 iSW[2]、iSW[1]和 iSW[0]作为信号输入端。输出引脚名为 oLEDG[0]，即可利用开发板上的一个绿色发光二极管 oLEDG[0]作为输出显示端。如何将此 4 个引脚和开关以及灯连接在一起呢？

对于 DE2-70 平台，拨动开关和发光二极管在硬件上是分别和 FPGA 的某些引脚连接在一起的，在使用的时候要参照 DE2-70 平台的引脚配置表，具体连接信息在 DE2-70 System CD-ROM 光盘的“DE2_70_pin_assignments. csv”文件中。例如 iSW[0]和 FPGA 的 pin_AA23 相连，oLEDG[0]和 FPGA 的 pin_W27 相连，只要将电路中定义的输入输出引脚指定到相应的 FPGA 外部引脚，也就和开发平台上相应的拨动开关以及发光二极管连接起来了。

打开“DE2_70_pin_assignments. csv”文件，查看外围部件和 FPGA 的引脚连接表，找到 iSW 和 oLEDG，根据引脚连接表配置引脚。

单击菜单项“Assignment→Pin Planner”打开引脚分配面板，在相应引脚的 Location 栏下双击，在弹出的下拉菜单中选择相应的 FPGA 引脚，如图 2-47 所示。

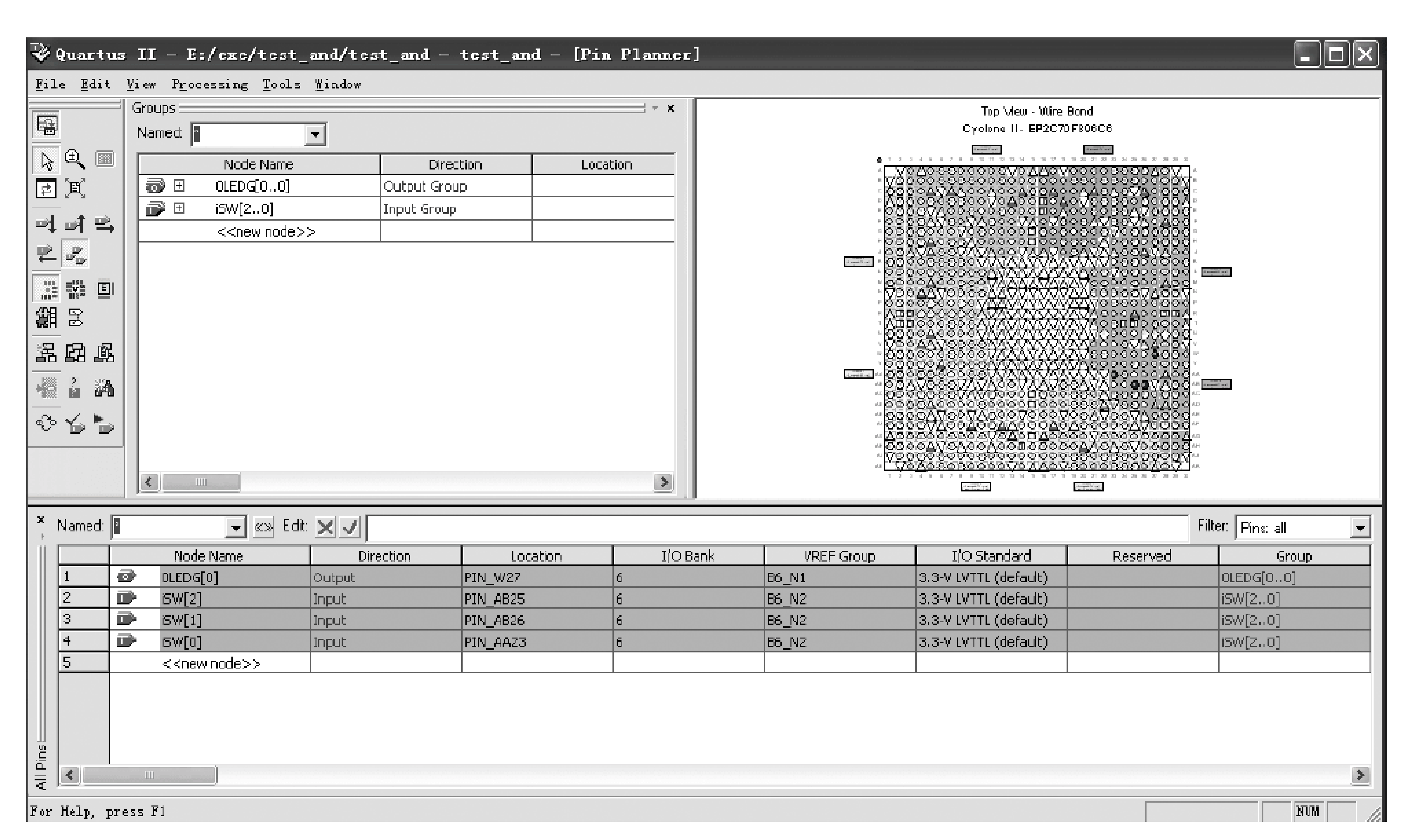


图 2-47　引脚分配界面

将未使用的管脚设置为三态。EP2C70F896C6 FPGA 共有 896 个管脚，本实验中只用了其中的 4 个，还有大量的管脚在本实验中并未使用，Quartus Ⅱ 默认这些未使用的管脚接地（逻辑 0），全部处于工作状态。这样 FPGA 工作起来的耗电量将增加，工作时间长

了后 FPGA 会发烫，这样将大大降低 FPGA 的工作寿命。因此，需要对所有未使用的管脚进行设置。

双击 Project Navigator 的 Hierarchy 标签栏中的 Cyclone Ⅱ：EP2C70F896C6，如图 2-48 所示，弹出如图 2-49 所示的界面。

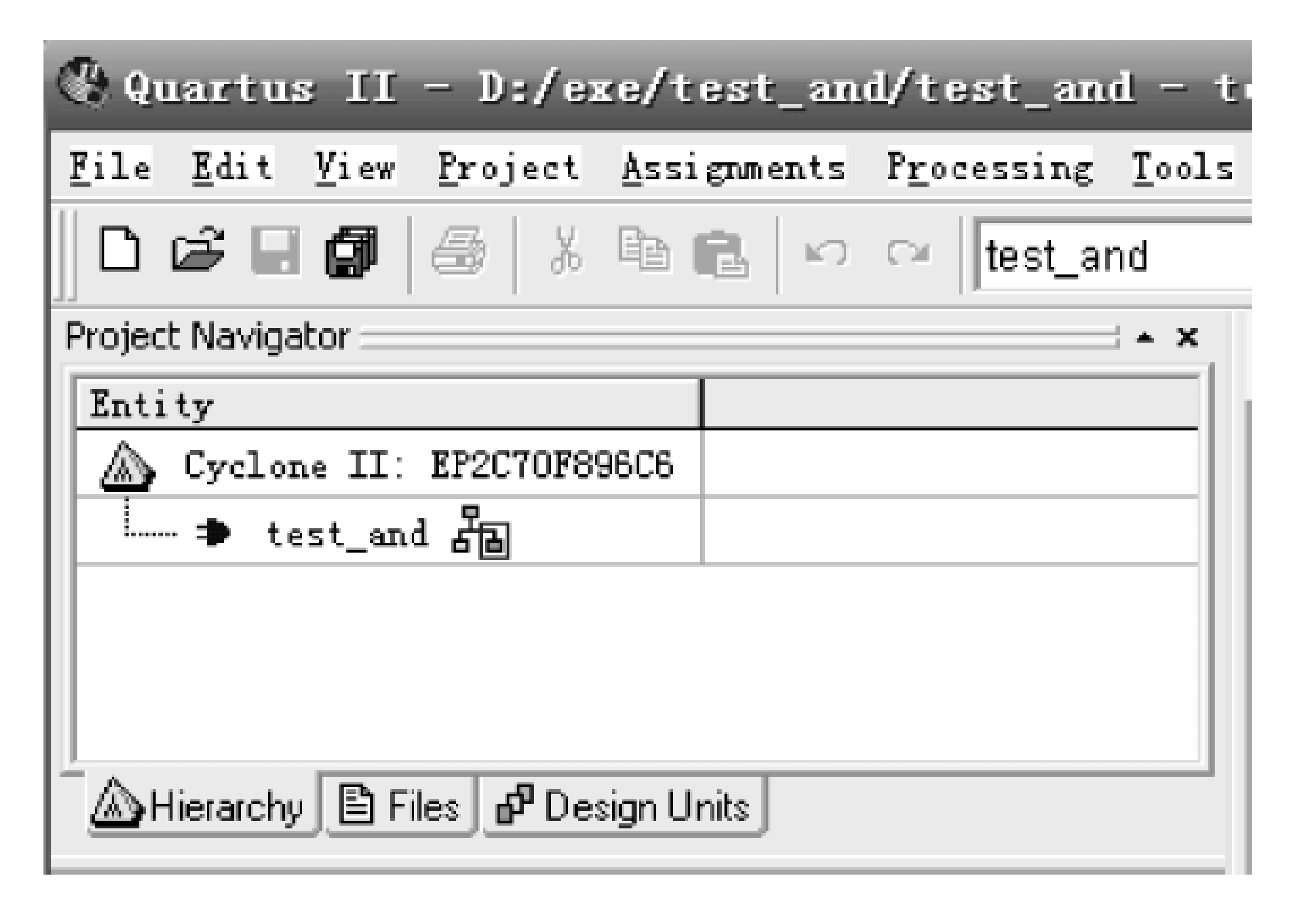


图 2-48　项目导航图

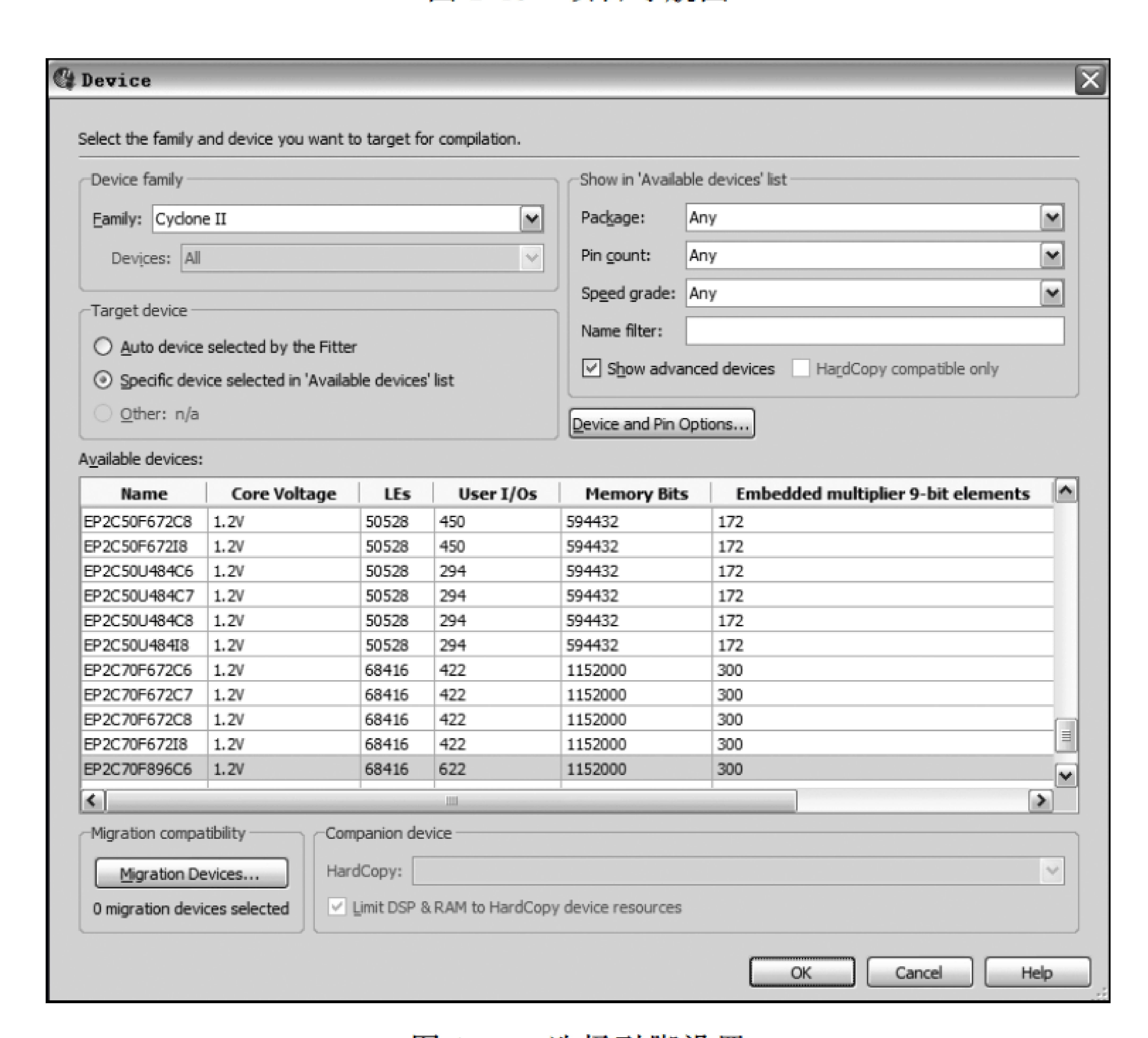


图 2-49　选择引脚设置

在图 2-49 中单击“Device and Pin Options”按钮，弹出如图 2-50 所示界面，打开“Unused Pins”选项卡，在“Reserve all unused pins”下拉框中选择“As input tri_stated”项，单击“OK”按钮，返回 Quartus Ⅱ界面。

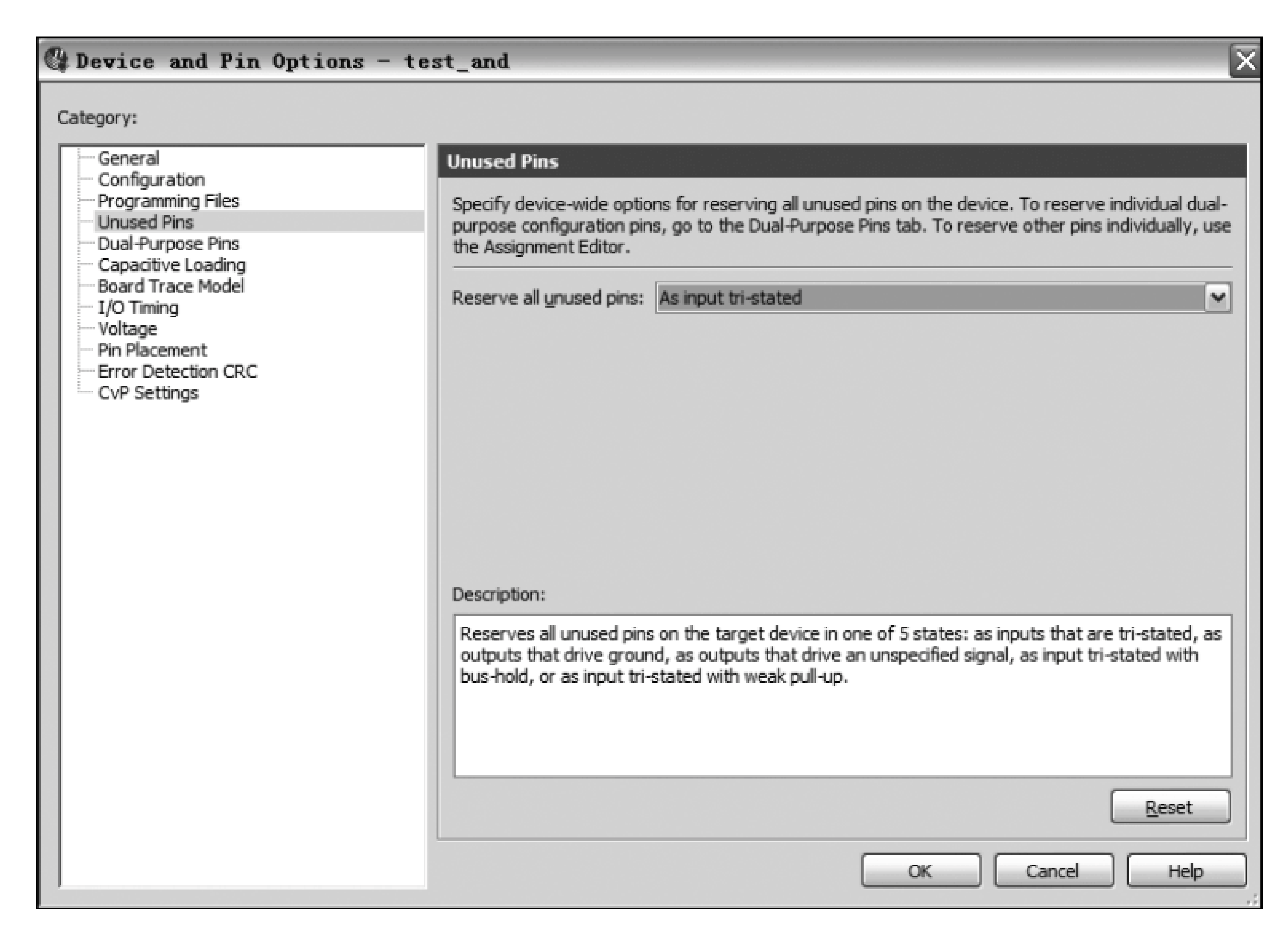


图 2-50　设置不使用引脚为三态

2.2.2.6　全编译文件

引脚分配完毕后，单击图标▶、单击菜单项“Processing→start compilation”或者使用快捷键 CTRL+L 执行全编译，生成用于配置 FPGA 的.sof 目标文件，如图 2-51 所示。

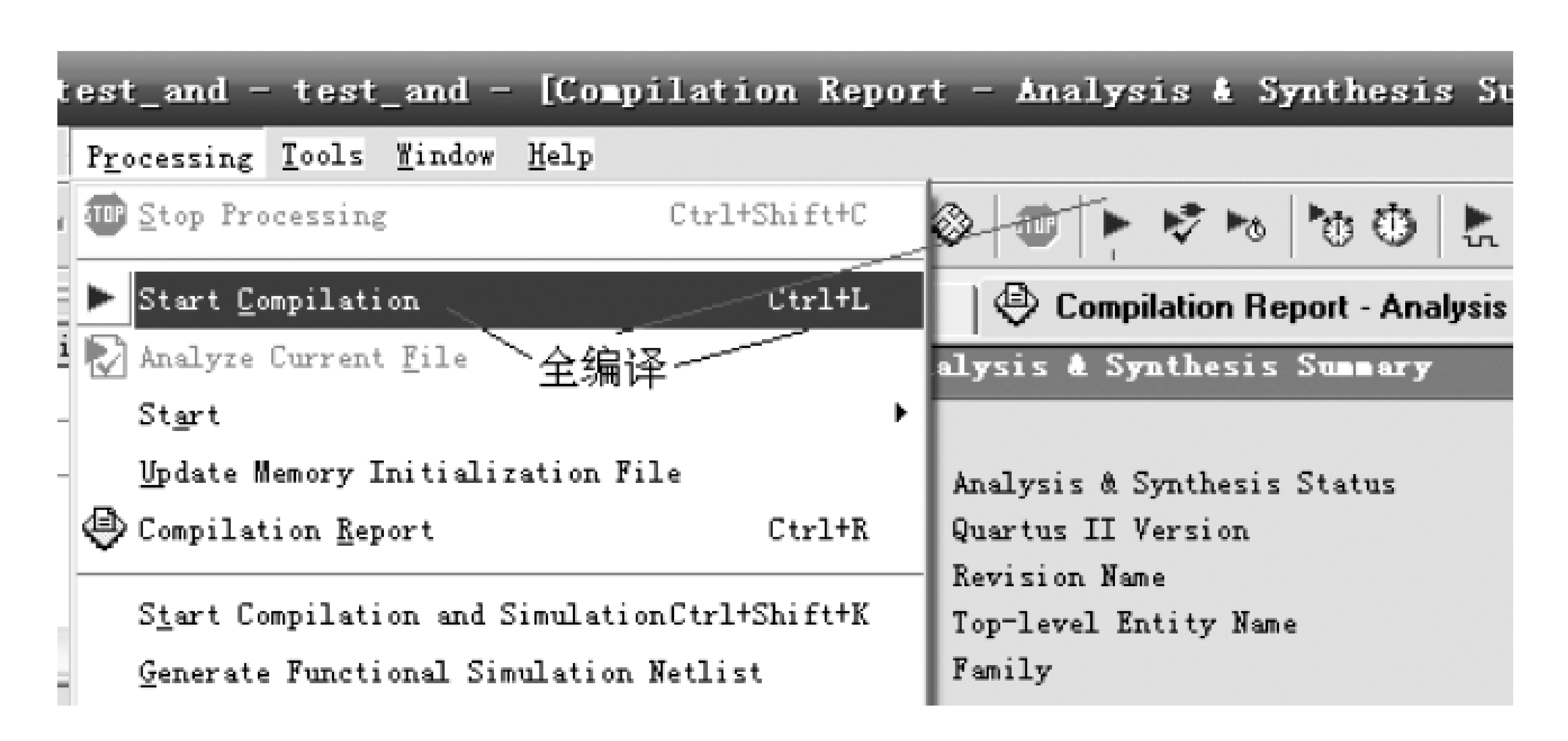


图 2-51　进行全编译

Quartus Ⅱ 软件包括模块化的编译器 Compiler，编译器包含以下模块，如图 2-52 所示。Analysis and Synthesis 完成分析与综合，Fitter 对设计进行布局布线，Assembler 针对目标芯片、电路逻辑和引脚配置情况产生用于下载到芯片上的编程文件，Timing Analyzer 分析和验证设计中的时序情况。

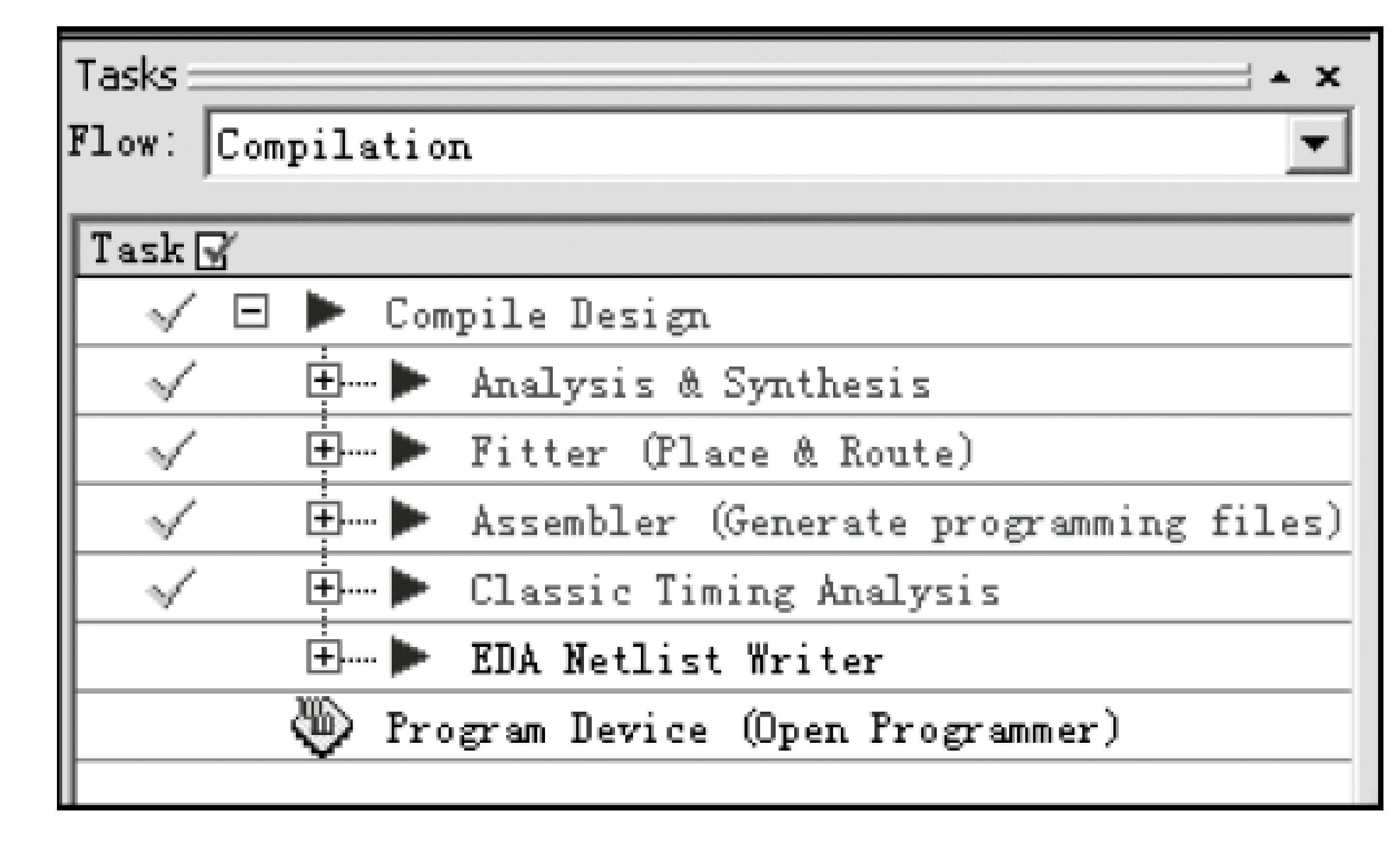


图 2-52　编译器包含的模块

全编译完成后，可以查看详细的编译

报告,如图 2-53 所示。再次请大家特别注意阅读全编译报告中的 Warning 信息,有些 Warning 信息无关紧要,有些 Warning 信息必须正确解决才能产生正确的结果,要在实践中总结经验,学会区别对待这些信息。

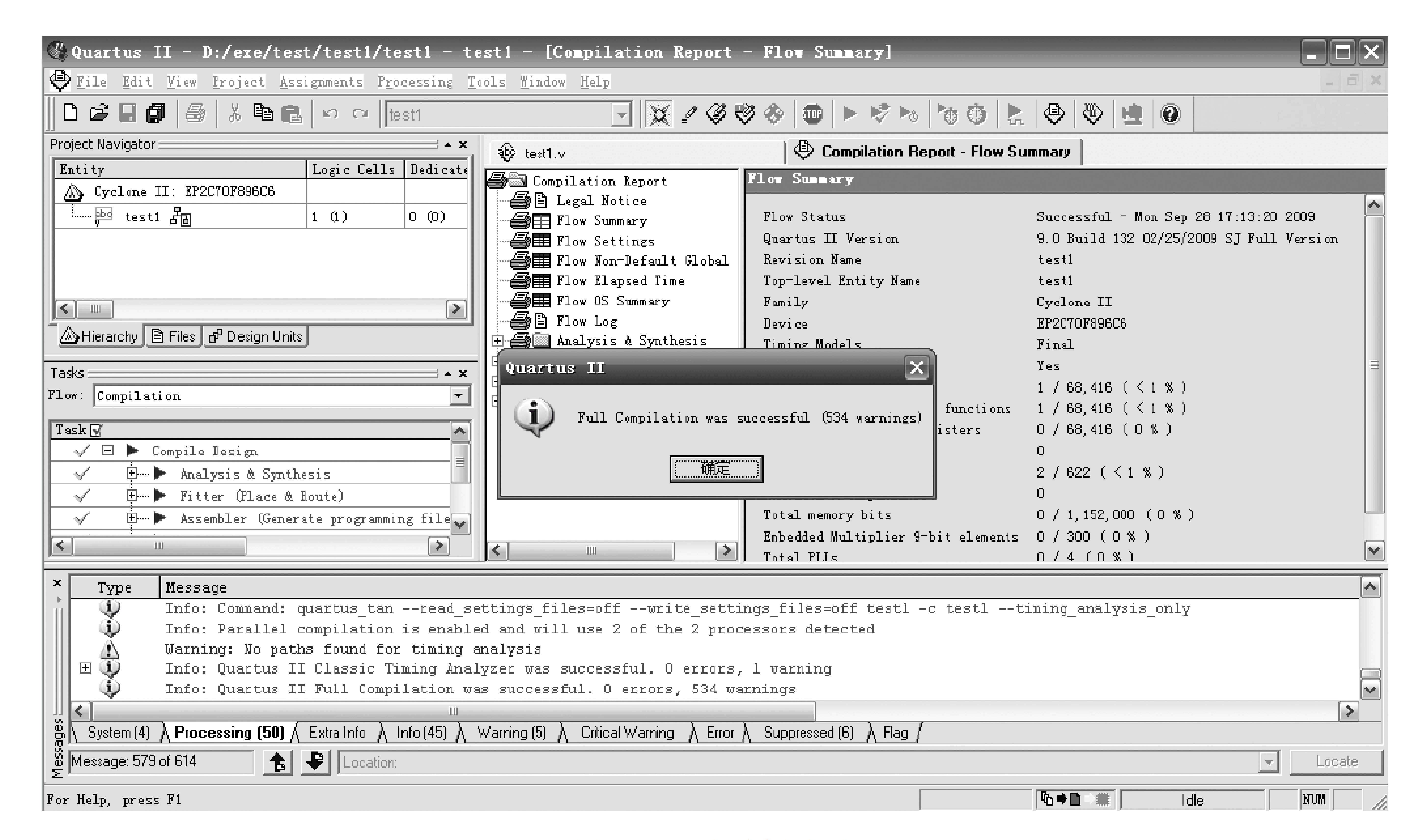


图 2-53 全编译成功

2.2.2.7 时序仿真

时序仿真,也称后仿真,用于查看理论上满足要求的设计,在具体的目标芯片中的时序是否满足项目要求。在全编译后,关闭功能仿真的 Modelsim 工作环境,单击按钮,重新启动时序仿真,在跳出的对话框“Timing model”栏中选择“Slow Model”,如图 2-54 所示。

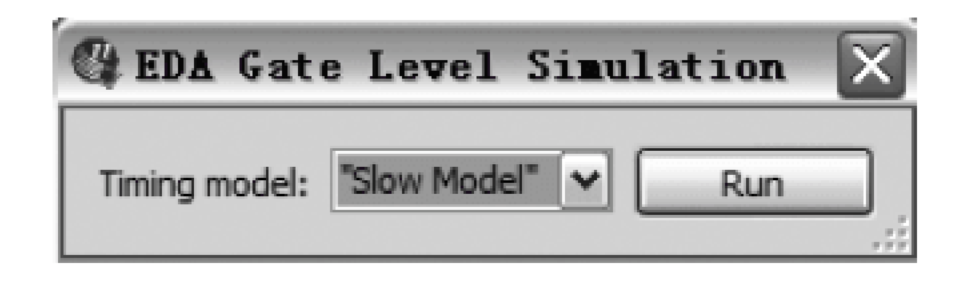


图 2-54 选择时序仿真

在时序仿真完成后,查看时序仿真结果。仿真结果显示:输入信号改变后延迟一段时间,输出信号才随之发生改变,如图 2-55 所示。在图中可以看出,器件每个门级的延时是纳秒(ns)级的,两级门的延时大概在几纳秒,因此信号改变的频率不可太快,否则无法分辨出某个时刻输出的变化是由哪个信号的变化引起的,一般要保持信号维持 40ns 以上的时间再变化一次。

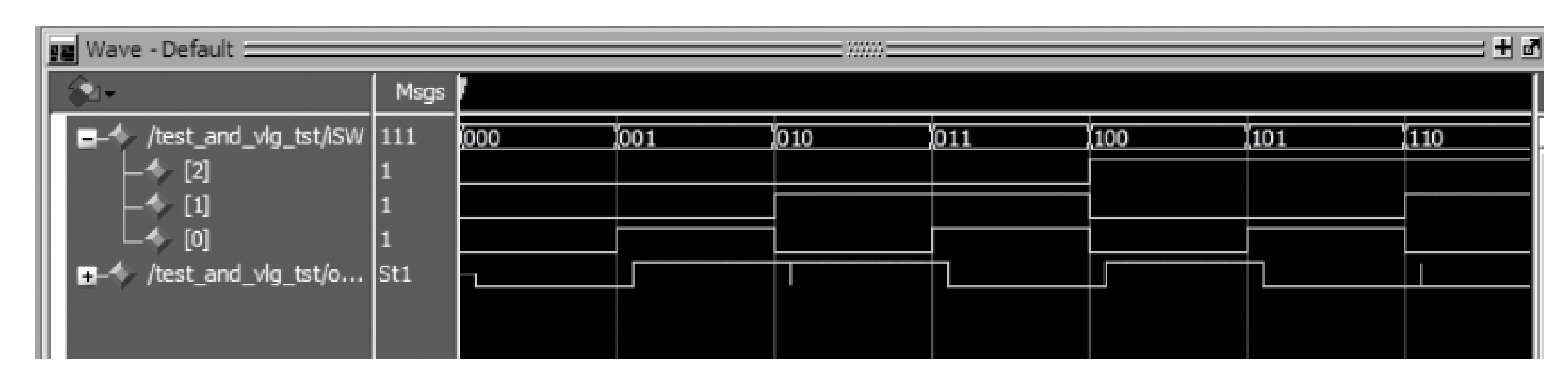


图 2-55 时序仿真结果

仔细观察图 2-55，图中出现了很多毛刺信号，这是因为在电路中有两个或者两个以上的输入信号同时改变时，每个信号到达同一个门的时间有长有短，这就是竞争现象，竞争的结果造成了短时间的输出结果与理论上的稳定输出结果的不同，产生了冒险现象。关于竞争和冒险现象分析请见附.1。

2.2.2.8 将设计下载到 FPGA 上进行验证

时序仿真满足设计要求之后，就可以将完成的设计下载到实验开发平台上进行验证了。单击图标或者单击菜单项“Tools→Programmer”，打开程序下载对话框，如图 2-56 所示。

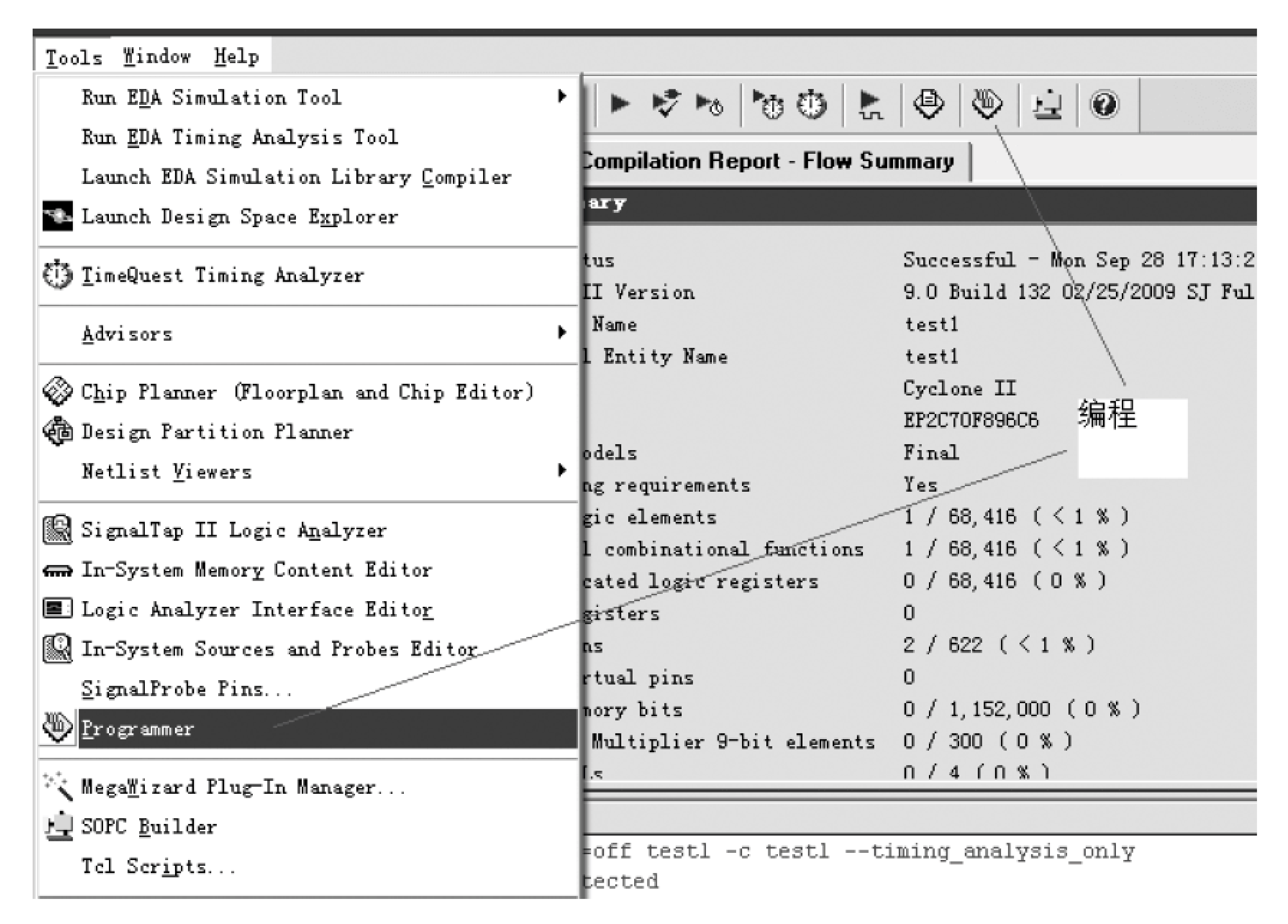


图 2-56　编程下载

显示编程界面，查看编程接口提示栏，如果显示信息为“No Hardware”，表明编程连接电缆没有设置好或者 USB-Blaster 的驱动程序没有安装，如图 2-57 所示。

检查计算机是否已经安装 USB-Blaster 的驱动程序，检查 DE2-70 开发板的 USB-Blaster 端是否和计算机正确相连，检查 DE2-70 开发板的电源是否接通，并且确定 DE2-70 开发平台的左侧中部的开关处在“RUN”位置。

注意：USB-Blaster 不能带电插拔。

连接好 USB-Blaster 后单击“Hardware Setup”选项，在“Currently selected hardware”的下拉菜单中选择“USB-Blaster [USB-0]”选项，如图 2-58 所示。单击“Close”按钮完成硬件设置。

如果“Currently selected hardware”的下拉菜单中没有“USB-Blaster [USB-0]”选项，则检查计算机的设备管理器，查看 USB-Blaster 的驱动程序是否安装。如果 USB-Blaster

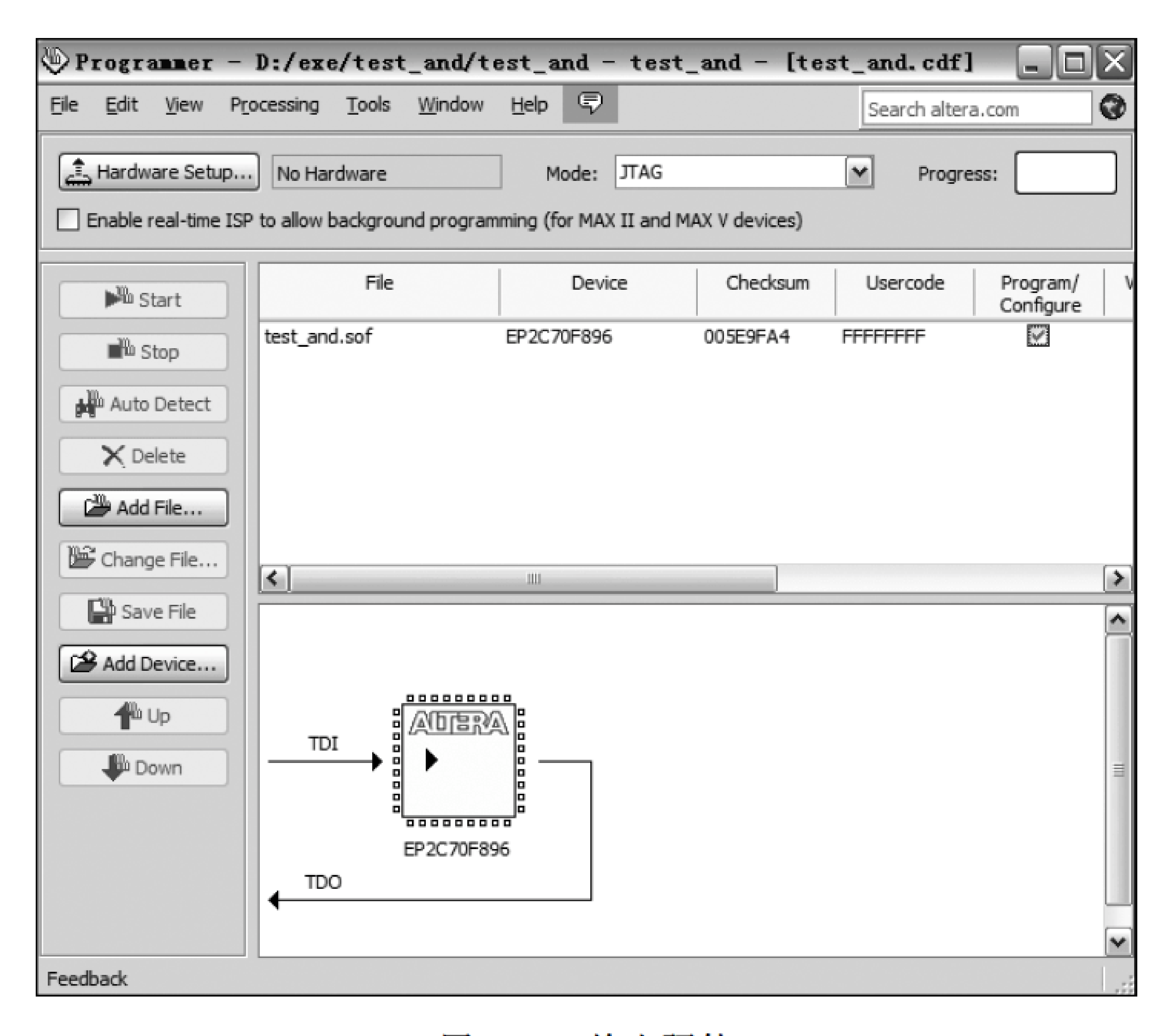


图 2-57　检查硬件

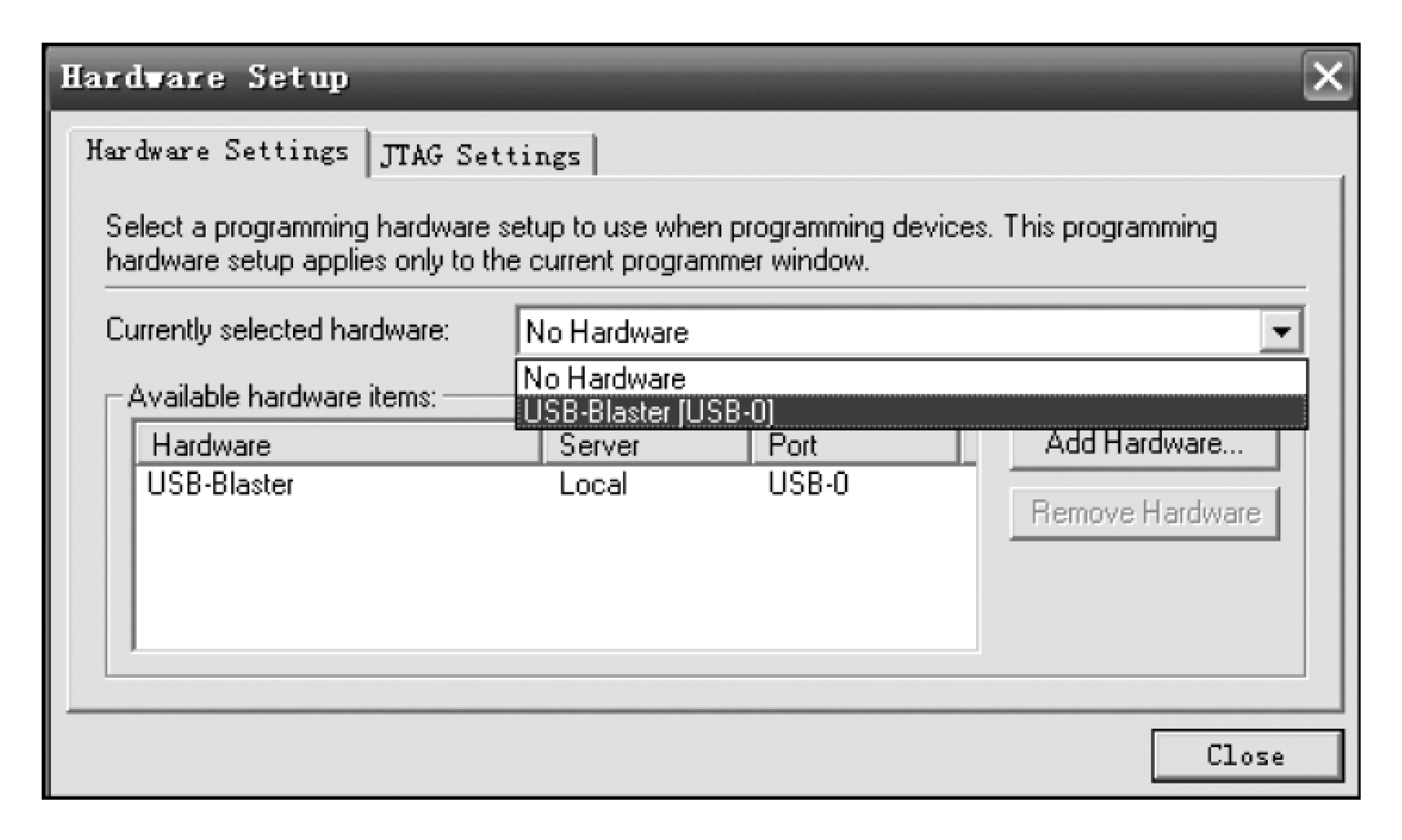


图 2-58　检查 USB-Blaster 是否安装

的驱动程序没有安装，在设备管理器下会有提示，如图 2-59 所示，在 USB-Blaster 上单击鼠标右键，更新驱动程序，Quartus Ⅱ的安装目录下有 USB-Blaster 的驱动程序。

设置好硬件接口的编程界面如图 2-60 所示。此时，编程文件“顶层实体名.sof”已经在文件列表中了，如果没有则利用“Add File”添加文件。“顶层实体名.sof”文件是编译器产生的数据文件，包含了 FPGA 的配置数据，如果单击“Add File”之后在工程目录下仍然无法找到“顶层实体名.sof”文件，则再次检查全编译时产生的 Warning 信息，查找错误产生的原因，最常见的原因是 Quartus Ⅱ的 license 设置不正确。



图 2-59　USB-Blaster 未安装提示

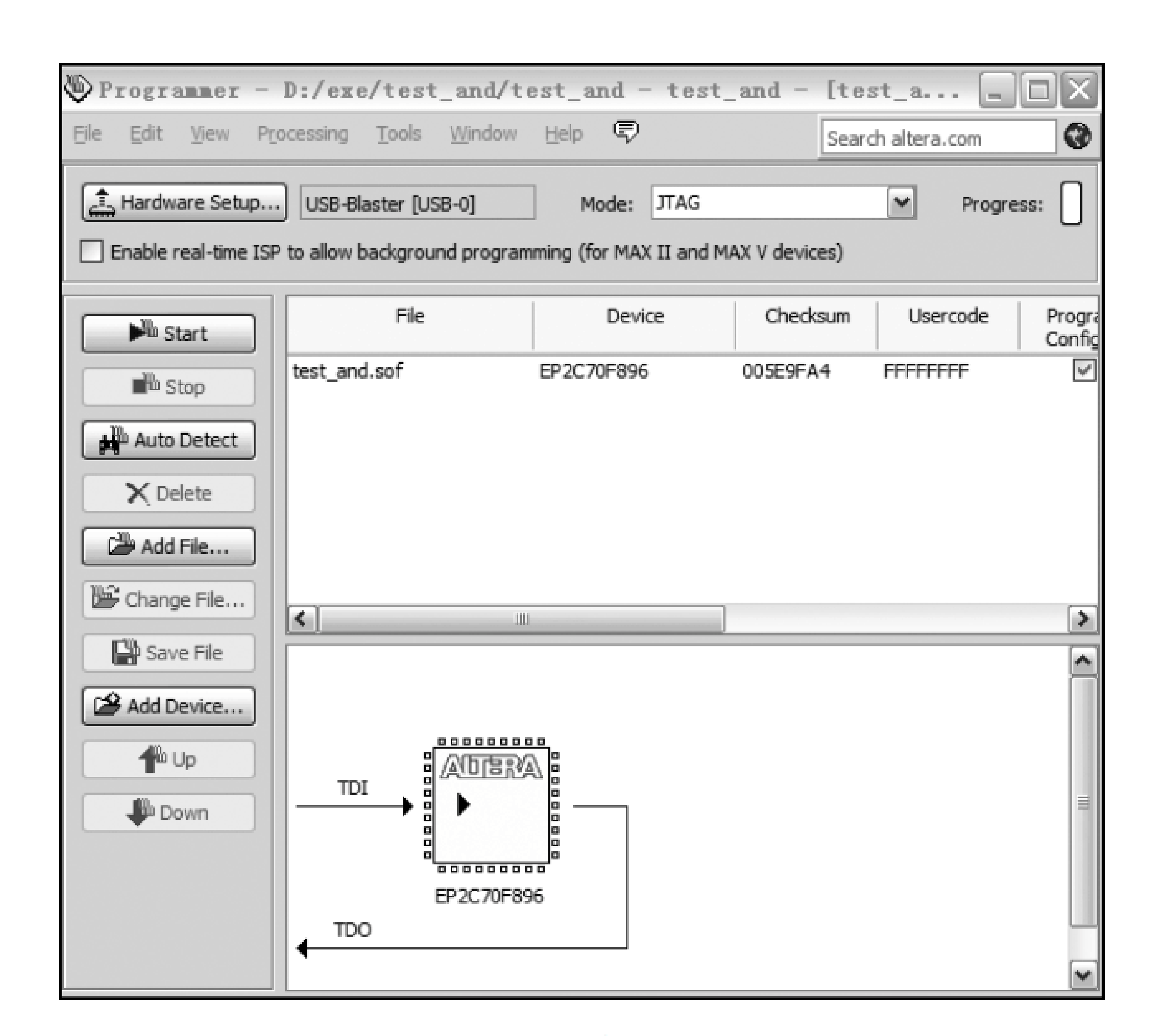


图 2-60　编程界面

单击"Start"按钮开始编程，如图 2-61 所示。如果 Quartus Ⅱ提示错误，检查电源及电缆连接是否正确。

编程结束后，编程进度条显示 100%，DE2-70 板上的发光二极管 LOAD 会闪烁一下。拨动开发板上的拨动开关 iSW[2]、iSW[1]和 iSW[0]，观察 oLEDG[0]是否按照设计要求显示，完成设计。

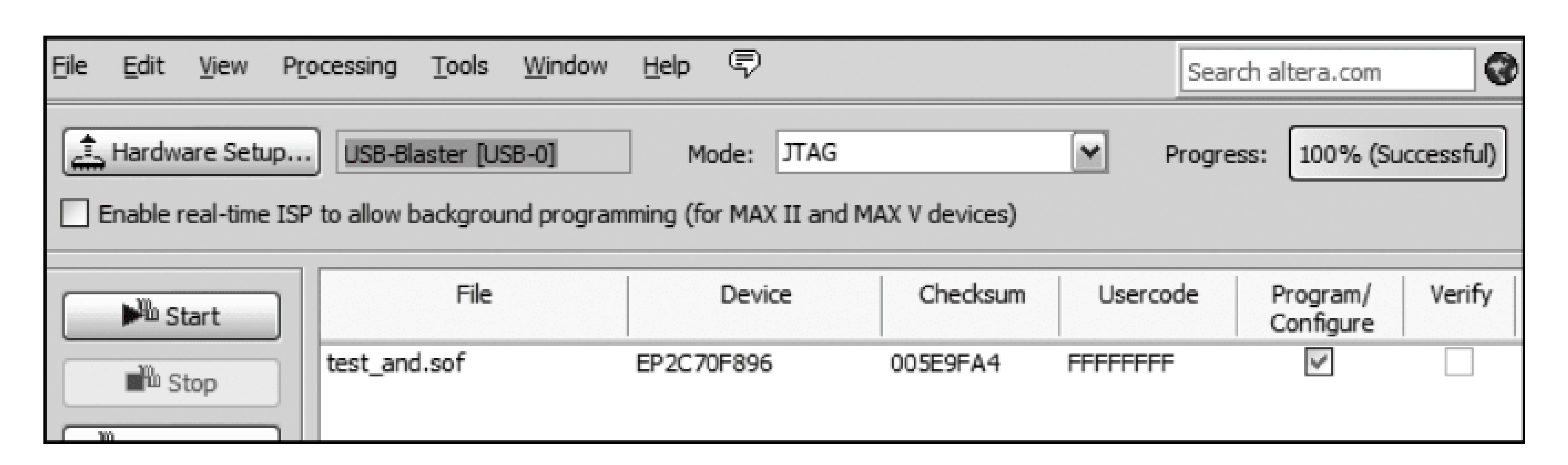


图 2-61 编程完成

2.2.3 尝试自己设计一个实验

可以尝试设计以下题目的实验：

(1) 假定一个楼梯，楼梯的中间有一个灯，在楼梯的最下面和最上面都有一个开关来控制这个灯，要求改变任何一个开关都能开灯或者关灯。

(2) 设计一个三人表决器，如果有两个或者两个以上的人同意，则结果为“同意”，否则为“不同意”。

(3) 设计一个电路，电路有两个输入端 $X(x_1、x_2)$ 和 $Y(y_1、y_2)$，如果 $X=Y$ 则输出为“1”，否则输出为“0”。

(4) 请观察日常生活中的现象，选择其中的一个，用一个简单的数字逻辑电路来实现这一现象，并且在 FPGA 上验证所设计的电路是否正确。

2.3 Verilog HDL 语言简介

要想利用 FPGA 的编译软件对设计的电路进行仿真，首先必须将我们的电路输入到编译软件中。在编译软件中输入电路主要有两种方法，一种是直接将电路原理图输入，另外一种方法就是采用硬件描述语言的方式。

硬件描述语言(Hardware Description Language，HDL)是一种用形式化方法来描述数字电路和设计数字逻辑系统的语言。它可以使数字逻辑电路设计者利用这种语言来描述自己的设计思想，然后利用电子设计自动化(EDA)工具进行仿真并自动综合到门级电路，再用 ASIC、CPLD 或 FPGA 实现其功能。目前，最主要的硬件描述语言是 VHDL 和 Verilog HDL，两者都符合 IEEE 标准。VHDL 发展得较早，语法严格，而 Verilog HDL 是在 C 语言的基础上发展起来的一种硬件描述语言，语法比较自由。目前，VHDL 和 Verilog 的应用都很广泛，在性能上也没有明显的优劣之分，学好了其中的任何一种语言都很容易过渡到另外一种语言。

本书使用的是 Verilog HDL 语言。本节将简单介绍 Verilog HDL 的概貌，有 C 语音基础的读者，在学习本节之后，将能够读懂 Verilog HDL 程序，如果想编写出风格良好的 Verilog HDL 程序，还要参考 *IEEE Standard Verilog® Hardware Description Language* 和其他相关书籍。

2.3.1 Verilog HDL 语言程序的结构

模块(module)是 Verilog 语言程序的基本单元，设计者设计的逻辑电路，无论规模大

小都定义在一个 module 中，每一个模块对应于一块硬件逻辑单元电路。

模块的基本结构如下：

```
module  模块名 (端口参数 1,端口参数 2,…);
        端口参数说明
        局部参数说明
        描述模块功能的语句
endmodule
```

模块以关键字 module 开始，每个模块都有一个名字，称为模块名。模块有输入参数和输出参数，这些称为端口。参数名和模块名的命名要求以字母开头，其中可以包含字母、数字、下划线和符号 $。另外，Verilog HDL 是对字母大小写敏感的，name1 和 Name1 表示两个不同的参数，建议在编写程序时不要用字母大小写的不同来区分不同的参数，因为如果将程序移植到字母大小写不敏感的环境中将会带来很大的麻烦。

模块要对端口参数，即输入输出参数进行说明，也要对模块内部的局部变量、常量和函数等进行说明，模块的局部参数对外是不可见的。

模块的主要部分是描述此电路要完成的功能，即 Verilog 语句。Verilog 语句可以指定模块在行为上的操作，例如赋值语句等；也可以实例化（即调用）其他模块。一个顶层模块可以实例化多个低层次的模块，多个顶层模块也可以实例化低层次的同一个模块，如图 2-62 所示。

Verilog HDL 的语言形式自由，不特别强调代码的形式，但是作为好的代码风格，代码应该编排得容易阅读，可以采用行缩进和空行来隔开代码的不同部分，使代码各个子功能模块一目了然，容易理解。代码中要适当地加入注释，注释可以有两种方式：一种是以双斜杠符“//”开始直至本行的末尾，另一种是所有包含在符号“/ * ”和“ * /”之间的部分。模块最后以关键字 endmodule 结束。Verilog HDL 语言要求，除了 endmodule 语句外，每条语句和数据定义的最后必须有分号。

考虑如图 2-63 所示的一个简单的 1 位比较器逻辑电路图。

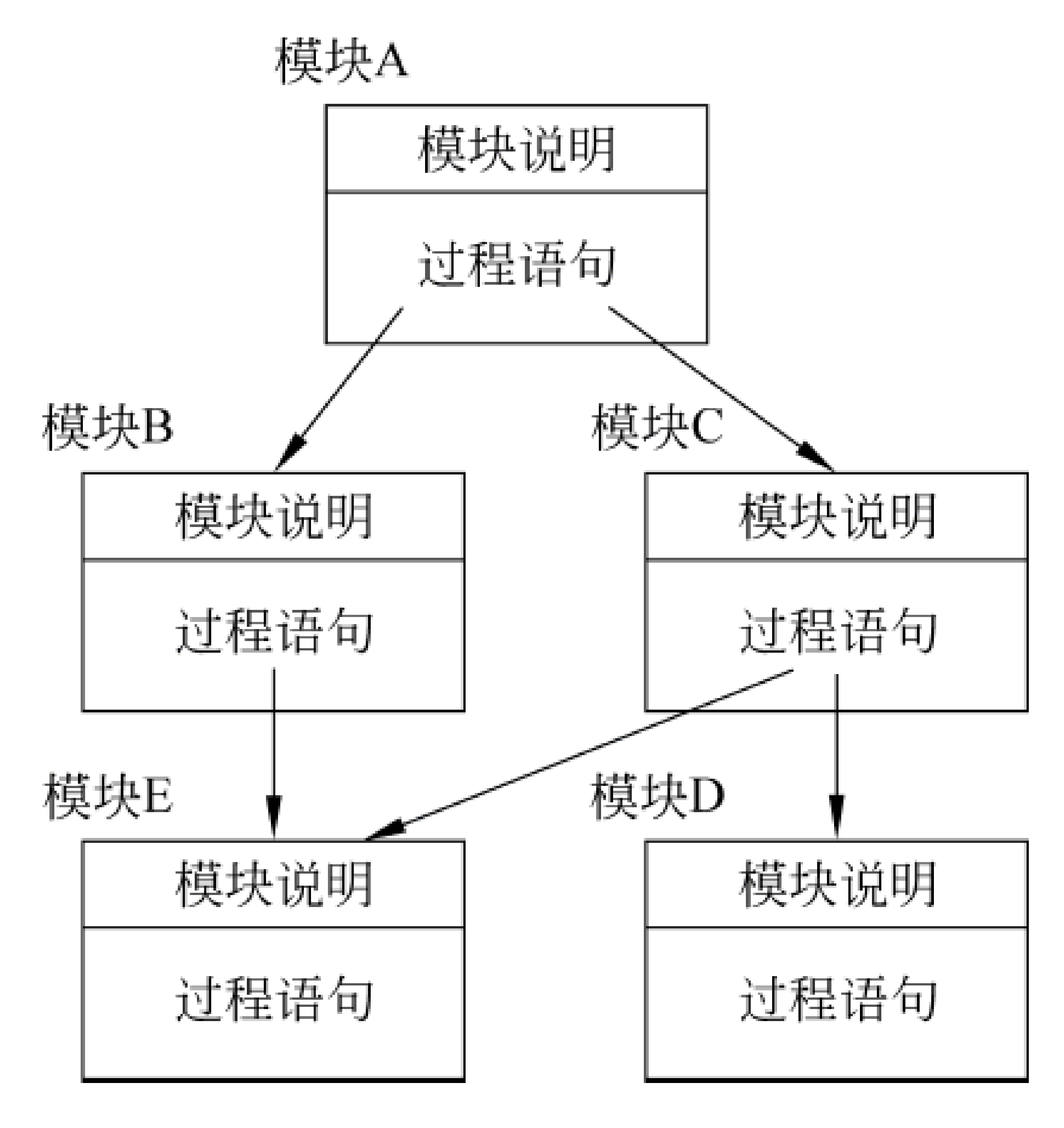


图 2-62　Verilog 模块实例化示意图

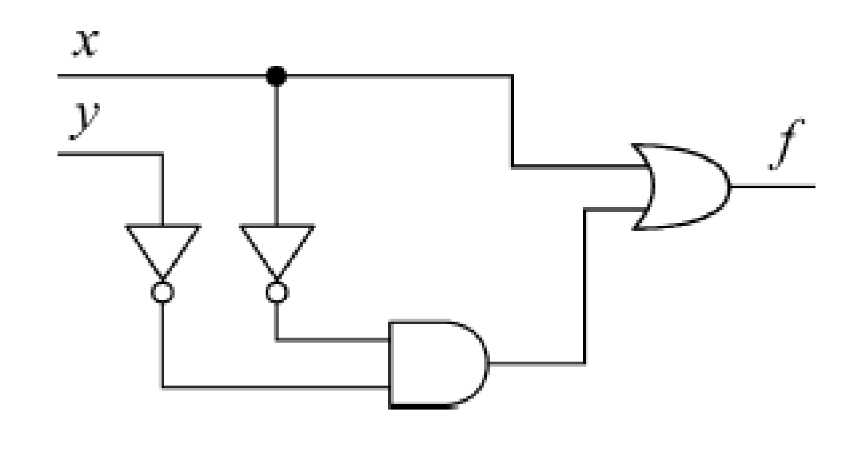


图 2-63　一个简单的逻辑电路

该简单的逻辑电路可用 Verilog HDL 代码表示如下：

```
module com_1(x,y,f);
        input   x,y;                    //输入参数
        output   f;                     //输出参数
                                        /* 模块要完成的逻辑功能 */
        assign   f=x|(~x & ~y);
endmodule
```

模块的名字是 com_1，模块中的前两条语句声明了该模块的输入输出端口，第三条语句描述该模块的具体电路结构或者电路功能。

2.3.2 逻辑系统、变量和常量

Verilog 使用四值逻辑系统——1 位码宽的信号可能的取值为以下 4 种之一：

0——逻辑“0”或假(false)；

1——逻辑“1”或真(true)；

x——未知状态；

z——高阻态。

Verilog HDL 中有十几种数据类型，其中最常用的是 wire 型、reg 型、integer 型和 parameter 型。

wire 型是 Verilog 数据线网类型(net type)中最常用的一种，它对应于电路元件中的物理连线，用于提供各模块和元件之间的连接。声明 Verilog 参数时，如果不指定数据类型，就默认为 wire 型。声明 wire 类型的格式如下：

```
wire[n-1:0] 参数 1,参数 2,…,参数 m;          //共有 m条总线,各总线的宽度是 n
wire[0:n-1] 参数 1,参数 2,…,参数 m;          //共有 m条总线,各总线的宽度是 n
wire[n:1] 参数 1,参数 2,…,参数 m;            //共有 m条总线,各总线的宽度是 n
```

例如：

```
wire    net_a;              //定义了一个 1位的 wire型参数 net_a
wire[7:0]   net_b, net_c;   //定义了两个 8位的 wire型参数 net_b和 net_c
wire[16:1]   net_d;         //定义了一个 16位的 wire型参数 net_d
```

reg 也是 Verilog HDL 中经常使用的变量类型，reg 类型的变量主要用于在 Verilog 程序中存储数值，并不是通常电路中的寄存器或触发器，reg 型的变量既可以用于时序逻辑电路的输出，也可以用于组合逻辑电路的输出。声明 reg 型变量的格式和声明 wire 类型的格式相同。例如：

```
reg   reg_a;                //定义了一个 1位的 reg型参数 reg_a
reg [7:0]   reg_b, reg_c;   //定义了两个 8位的 reg型参数 reg_b和 reg_c
reg [16:1]   reg_d;         //定义了一个 16位的 reg型参数 reg_d
```

integer 型参数也是 Verilog HDL 中经常使用的一种变量类型，它的值是一个 32 位或者更长的整数（取决于模拟器所采用的字长）。integer 变量一般在 Verilog 程序中控制重复语句的执行（如 for 循环语句），其声明格式如下：

```
integer    integer_a,integer_b;          //定义了两个整型参数 integer_a和 integer_b
```

数组类型也称为 memory 型，Verilog HDL 通过对 reg 型参数建立数组来定义存储器，数组中的每一个单元通过一个数组索引进行寻址。memory 型的参数声明格式如下：

```
reg [n-1:0]   mem _a[m-1:0];
reg   mem _b[m-1:0];
```

对于第一个声明格式，reg[n－1:0]定义了存储器中每一个存储单元的大小，即每一个单元是一个 n 位的寄存器；而 mem_a[m－1:0]则定义了 m 个这样的寄存器。对于第二个声明格式则表明定义了 m 个 1 位的寄存器。

例如：

```
reg [8-1:0]   mem _a[5:0],mem _b[255:0];
```

这里定义了两个 memory，mem _a 是一个含有 6 个 8 位寄存器的 memory；mem _b 是一个含有 256 个 8 位寄存器的 memory。

常量是在程序运行过程中不被改变的量，Verilog HDL 语言中的常量有 4 种不同进制的表现形式：二进制数（b 或 B）、八进制数（o 或 O）、十进制数（d 或 D）和十六进制数（h 或 H），数字的表达方式为：

```
<位宽>'<进制><数字>;
```

这是一种完整的表示方式，其中<位宽>和<进制>都可以缺省，其中的位宽是指此数字在机器中以二进制表示的位数，并不是此数字在指定进制中的位数。缺省位宽，则采用默认位宽，默认位宽同机器字长；缺省进制值则默认为十进制。

例如：

```
10'b1010101010            //10 位二进制数 1010101010
12'dz                     //12 位十进制数，其值为高阻态
8'hFx                     //8 位十六进制数，其高位值为 F，低位值为不定值
```

Verilog HDL 用 parameter 来定义常量，这样可以提供代码的可读性和可维护性。用 parameter 定义常量的说明格式如下：

```
parameter   a=表达式 1,b=表达式 2;
```

例如：

```
parameter   a=16,b=37;        //定义了两个常量参数 a=16 和 b=37
```

如果一个标识符被赋给一个常量值，那么在整个当前模块中这个标识符就表示这个常量值，而且有效范围仅限于定义的这个模块中。

2.3.3 操作符和表达式

Verilog HDL 中变量的值是通过表达式来改变的，表达式通过赋值语句将值赋给变量。如果表达式结果的位数比变量小，那么就要在变量的左边补 0；如果表达式结果的位数比变量的位数大，则保留结果的右边数值。

表达式是由常量、变量和操作符构成的，常量和变量在上一小节中有所介绍，本节介绍 Verilog HDL 中常用的操作符。

2.3.3.1 布尔操作符

布尔操作符如表 2-2 所示，也称为按位操作符，其功能是将操作符两边的操作数按位(低位对齐，高位补 0 对齐)进行逻辑运算。在布尔操作中要注意当输入有 x 和 z 值时的输出值，如果输入中有一个或者两个输入信号是 x 或 z，除非输出信号受到 z 的控制，否则输出值要仔细考虑 x 值。

表 2-2 布尔操作符

操作符	含义	操作符	含义
&	与	~^,^~	异或非(同或)
|	或	~	非
^	异或		

2.3.3.2 算术和移位操作符

算术和移位操作符如表 2-3 所示，它把向量当作无符号数来处理。加法和减法在 Verilog 综合工具中一般被综合成加法器和减法器，乘法也会综合出乘法器，但是效率不一定高，除法和取模运算效率很低，在运算中尽量少使用。对于除数是 2 的幂的操作，可以利用向右移位操作(除法)，或者选取被除数的最右边位来完成(取模)。

表 2-3 算术和移位操作符

操作符	含义	操作符	含义
+	加	%	取模
−	减	<<	向左移位
*	乘	>>	向右移位
/	除		

移位操作符的第二个操作数是需要移位的位数，移位后空出的位补 0。

2.3.3.3 逻辑操作符

Verilog 中的逻辑操作符如表 2-4 所示，逻辑操作符的运算结果值是真/假。

在 Verilog 中，1′b1 的值被认为是“真”(true)，而 1′b0 的值被认为是“假”(false)。对于多位值，只有此值的各位都为 0，此值才为假。逻辑操作的结果也是 1 位的值：“1”

(真)/“0”(假)。

表 2-4　逻辑操作符

操　作　符	含　　义	操　作　符	含　　义
&&	逻辑与	>	大于
||	逻辑或	>=	大于或等于
!	逻辑非	<	小于
==	逻辑相等	<=	小于或等于
!=	逻辑不相等		

2.3.3.4　条件操作符(?)

条件操作符表达式形式为:

```
X?Y:Z
```

当 X 值为真时,该语句的值为 Y,否则为 Z。

2.3.3.5　连接操作符({})

当需要将若干个向量或者位合并成新的向量时,可以使用连接操作符:

```
{x,…,y}
```

将 x,…,y 各向量首尾相接。

```
{n{x}}
```

将 x 重复 n 次。

2.3.4　电路设计的三种不同形式

传统编程语言中的各语句是按照语句排列的先后“顺序”执行的,而 Verilog HDL 模块中的语句是要模拟硬件的操作行为,硬件中的各元件是按照设计好的时间步骤彼此连接相互作用的,所以 Verilog HDL 模块中的语句是一系列“并发”执行的语句,即 Verilog 中的每个并发语句和同一个模块中的其他语句“同时”执行。Verilog HDL 模块基本的几种语句是实例语句、连续赋值语句和 always 程序段,本节我们将介绍这几种并发语句。

2.3.4.1　逻辑电路的结构形式描述

Verilog 中包含了一套常用逻辑门的门级原语(gate level primitive)。逻辑门由它的函数(功能)名和输出输入参数表示,这些门的名字都是保留字。例如,and(与)、or(或)和 xor(异或)门以及它们的补,还有 not(非)门等。这些内置的门都可以有任意的输入端数,它们定义的端口顺序应该是:输出、输入、输入、…、多个输入端之间的顺序无所谓。对于有使能端的内置门,其端口顺序应该为:输出、数据输入、使能输入。

典型的 Verilog 设计环境还包括一些库,提供许多常用的预定义组件,例如,与或非

门、触发器以及一些其他的功能部件。另外，用户在设计过程中也会定义一些组件。使用这些门和其他组件时，要用实例语句将其实例化。

在结构形式的电路描述或设计中，各个组件和门都被实例化，彼此间通过网格变量相连。结构化的电路描述等效于用语言的形式实现电路原理图。

实例化组件时，首先要给出组件的名字(如 and)，跟着是该实例的名字(可选)，最后是用括号括起来的，与组件端口相对应的输入输出列表。

考虑图 2-64 的一个简单的逻辑函数电路图，用 Verilog HDL 结构化描述方式的代码表示如下。

```
module mux2to1_1(x1,x2,x3,f);
    input  x1,x2,x3;          //输入参数
    output  f;                //输出参数
    wire g,k,h;
        /* 模块要完成的逻辑功能 */
    and (g,x1,x2);
    not (k,x2);
    and (h,k,x3);
    or (f,g,h);
endmodule
```

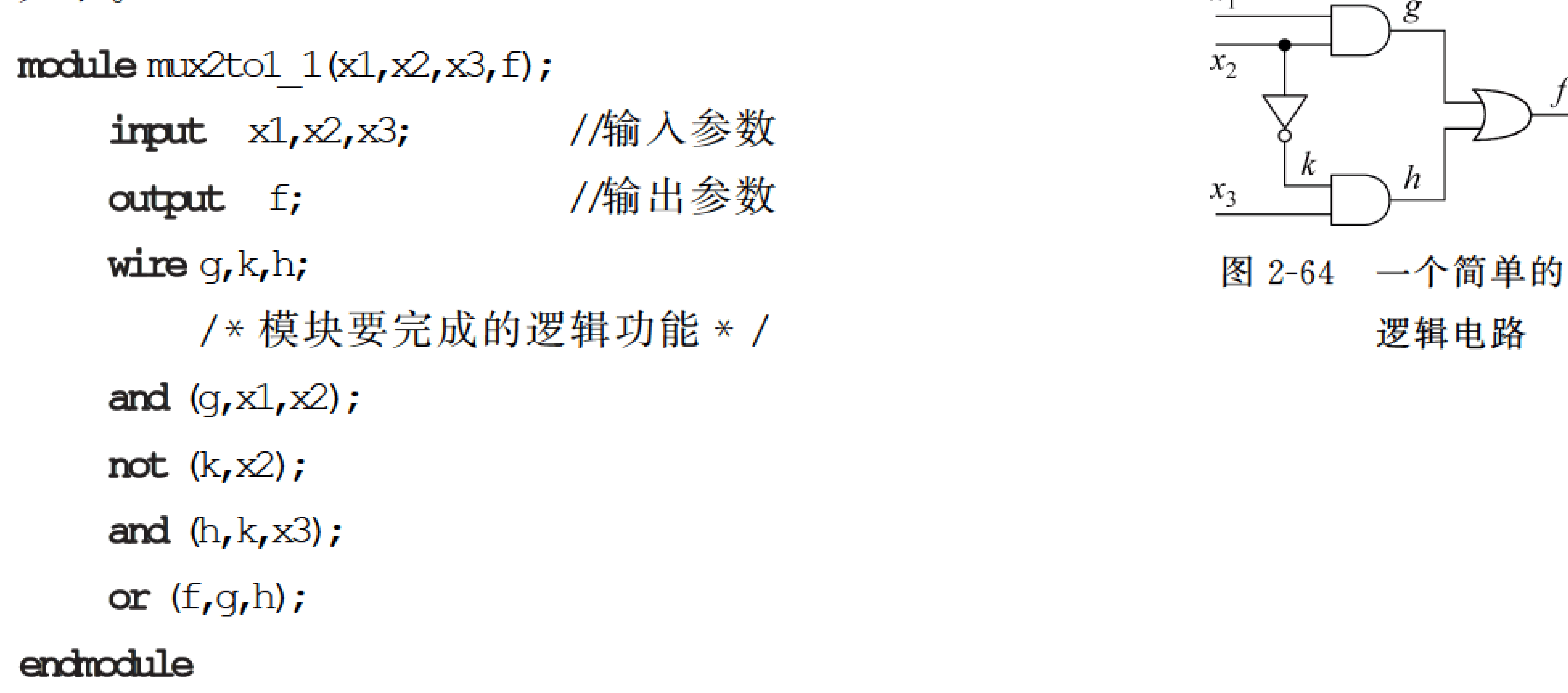


图 2-64 一个简单的逻辑电路

2.3.4.2 逻辑电路的数据流形式描述

采用门级原语设计简单逻辑电路直观易懂，但是如果设计大型的数字电路采用门级原语将会非常麻烦。Verilog HDL 语言还允许根据数据流程和电路的操作来设计逻辑电路，这种描述形式被称为数据流形式的电路设计。图 2-64 电路中所示的电路逻辑功能可表示为：

$$f = (x_1 x_2 + \bar{x}_2 x_3)$$

用 Verilog 代码表示如下：

```
module  mux2to1_2(x1,x2,x3,f);
    input  x1,x2,x3;
    output f;
/* 用连续赋值语句进行功能定义 */
    assign  f= (x1 & x2) | (~ x2 & x3);
endmodule
```

“&”、“|”和“~”符号分别表示“与”、“或”和“非”操作，assign 关键字为信号 f 提供连续赋值，等式右边的值被不停计算，结果赋给 f。assign 称为连续赋值语句，连续赋值语句是并发执行的，各语句的执行次序与其在描述中出现的次序无关。assign 语句是 Verilog HDL 语言的基本赋值语句，数字系统中凡是直接以线网方式连接的结构，都要采用 assign 语句赋值。

2.3.4.3 逻辑电路的行为风格描述

Verilog HDL 还有一种设计逻辑电路的方法，考虑图 2-64 所示的电路，电路所描述的是一个二选一的选择器，其中 x_2 为控制端，x_1 和 x_3 为数据输入端，当 $x_2=0$ 时输出 $f=x_1$，当 $x_2=0$ 时输出 $f=x_3$。我们采用如下 Verilog HDL 代码完成电路设计：

```
module mux2to1_3(x1,x2,x3,f);
    input  x1,x2,x3;
    output f;
    reg  f;
    /* 在 always 结构块中完成电路功能 */
    always @ (x1 or x2 or x3)
        if (x2==1)
            f=x1;
        else f=x3;
endmodule
```

代码中的 if-else 语句是 Verilog HDL 语言中的一种过程性语句，Verilog 语法要求过程性语句必须包含在叫做 always 的模块中。always 模块@符号后面圆括号里的部分，称为敏感列表，一旦敏感列表中的任意一个值发生变化，always 模块中的语句就会执行，对模块中的 f 变量进行赋值，这个值将一直“寄存”在 f 中，直至下一次敏感列表中的参数发生变化，f 的值才会被重新赋值。因此，变量 f 必须被声明为 reg 型的变量。

必须注意的是，用 Verilog HDL 实现一定的功能时，其模块中的逻辑功能是并发执行的。也就是说，如果在一个模块中同时出现了门级原语、assign 语句和 always 模块，它们的次序不会影响逻辑实现的功能，这三项是并列执行的。然而，在 always 模块内，逻辑是按照指定的顺序执行的，看一下 always 内的语句，你就会明白它是如何实现功能的：if-else，if 必须先执行，否则其功能就没有任何意义。另外，两个或更多的 always 模块也是同时执行的，但是 always 模块内部的语句是顺序执行的，因此 always 模块中的语句称为“顺序语句”。

这一描述方式描述了电路要完成的功能，即电路的行为，我们称之为行为级描述。

在模块中，各种描述风格可以自由混合使用。

特别提醒：Verilog HDL 语言内容丰富，功能强大，但是建议读者要选用自己熟悉的语法来描述电路，大多数硬件开发工程师的经验表明，Verilog HDL 语言在熟练使用时仅仅需要极少数常用的语句。特别要提醒的是，请严格区分 C 语言和 Verilog HDL 语言，不要以 C 语言的编程习惯来使用 Verilog HDL 语言，Verilog HDL 语言是一种硬件描述语言，写代码前要首先建立硬件的模型，然后再利用 Verilog HDL 语言将这个硬件模型描述出来。

第 3 章 组合逻辑电路设计

组合逻辑电路是数字系统中最基础的电路，它在任一时刻的稳态输出只取决于该时刻的输入变量值，而与该电路以前的输入变量值无关。在组合逻辑电路中可以包含任意多的逻辑门和反向器，但是不含有反馈电路，反馈电路会产生时序电路的特性。

3.1 选择器实验

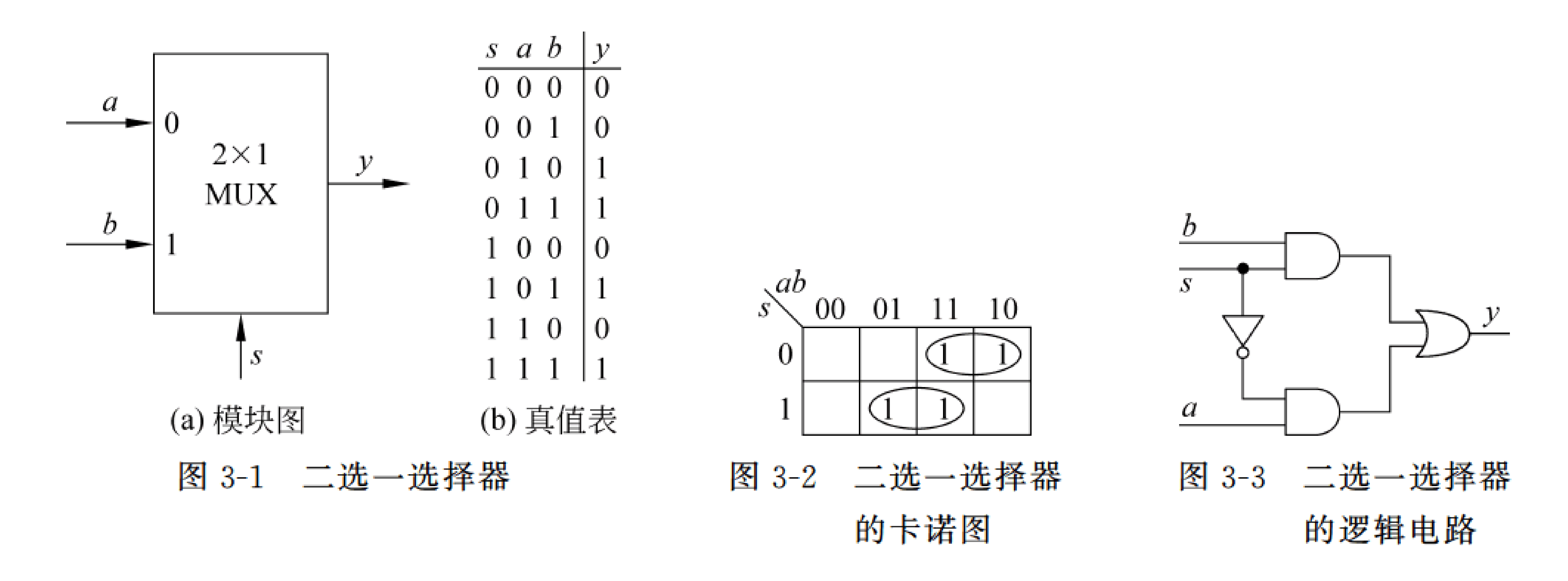
选择器是数字逻辑系统的常用电路，是组合逻辑电路中的主要组成元件之一，它是由几路数据输入、一位或多位的选择控制端，以及一路数据输出所组成的。多路选择器从多路输入中，选取其中的一路将其传送到输出端，由选择控制信号决定输出的是第几路输入信号。

本次实验介绍几种常用的多路选择器的设计方法；Verilog 语言中的 if-else 语句和 case 语句的使用；在 Quartus Ⅱ 中，利用导入引脚文件的方式来配置引脚的方法等。最后，请读者自行设计一个多路选择器，熟悉电路设计的基本流程和 Quartus Ⅱ 的使用。

3.1.1 二选一多路选择器

图 3-1 是二选一选择器的模块图和真值表，图中 a 和 b 为输入端；y 为输出端；s 是选择端，选择两个输入的其中一个输出。当 s 为 0 时，y 的输出值为 a；当 s 为 1 时，y 的输出值为 b。

图 3-2 是二选一选择器的卡诺图，根据卡诺图得出二选一选择器的表达式为 $y=(\sim s\&a)|(s\&b)$。根据表达式画出其逻辑电路如图 3-3 所示。



s	a	b	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

s \ ab	00	01	11	10
0			1	1
1		1	1	

(a) 模块图　(b) 真值表

图 3-1　二选一选择器

图 3-2　二选一选择器的卡诺图

图 3-3　二选一选择器的逻辑电路

利用 Verilog HDL 实现二选一选择器的逻辑电路，如程序清单 3-1 所示。

程序清单 3-1　mux21.v

```
module mux21(a,b,s,y);
input   a,b,s;
output  y;
assign  y=(~s&a)|(s&b);
endmodule
```

上述代码分析与综合后的仿真结果如图 3-4 所示，由图中可以看出，当 $s=0$ 时，$y=a$，即 y 随着 a 值的改变而改变，此时的 b 值无论如何改变都不影响 y 的值。当 $s=1$ 时，$y=b$，即 y 随着 b 值的改变而改变，此时的 a 值无论如何改变都不影响 y 的值。

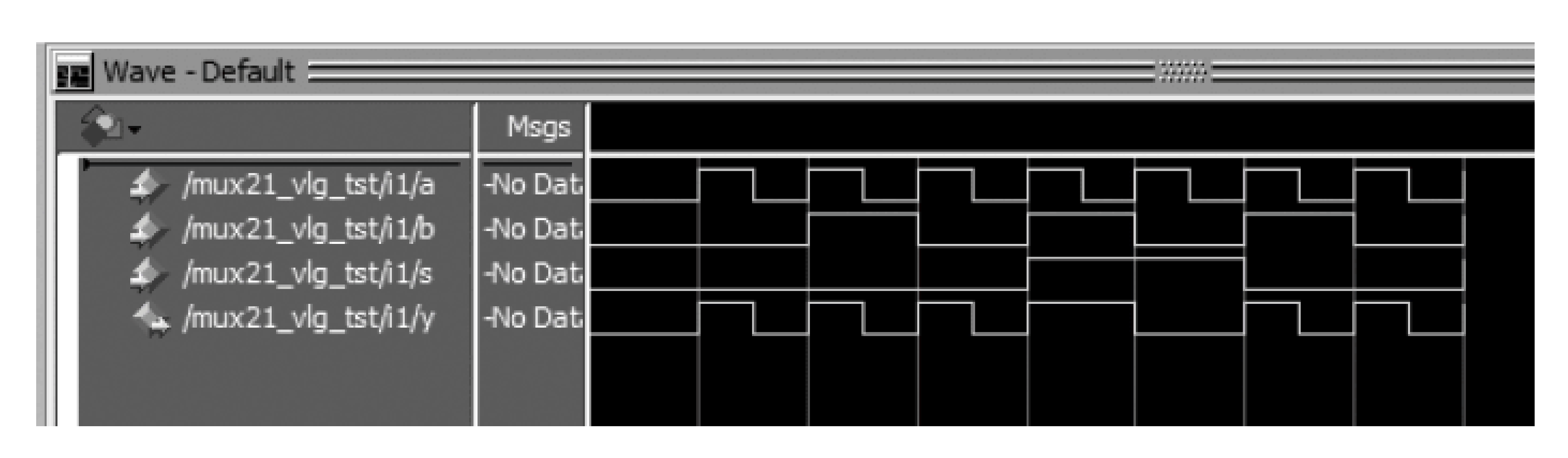


图 3-4　二选一选择器 Verilog 程序仿真结果

3.1.2　四选一多路选择器

四选一多路选择器的模块图和真值表如图 3-5 所示，$a_0 \sim a_3$ 为 4 个输入端，s_0 和 s_1 是选择端，y 是输出端，根据 s_0 和 s_1 值的不同，y 选择 $a_0 \sim a_3$ 中的一个输出，具体请见真值表。

在 Verilog 语言中的 case 语句可以综合出“多路复用器”的电路，它的可读性非常强。程序清单 3-2 所示的是用 case 语句实现四选一多路选择器的方法。

a_0　a_1　a_2　a_3　0　1　2　3　4×1 MUX　y　s_0　s_1

(a) 模块图

s_1	s_0	y
0	0	a_0
0	1	a_1
1	0	a_2
1	1	a_3

(b) 真值表

图 3-5　四选一选择器

程序清单 3-2　mux41.v

```
module mux41(a,s,y);
input   [3:0] a;
input   [1:0] s;
output reg y;
always @ (s or a)
case  (s)
        0: y=a[0];
        1: y=a[1];
        2: y=a[2];
        3: y=a[3];
        default: y=1'b0;
endcase
```

```
endmodule
```

case 语句是以关键字 case 和一个被括起来的“选择表达式”开头，表达式的结果表示一个整数。下面是 case 选项，每个选项由选择列表和过程语句构成，选择列表可以是一个整数值，也可以是多个整数值，多个整数值之间以逗号分开，选择列表和过程语句之间以冒号连接，例如 0,1：y=a[0]；。

case 语句的执行过程是这样的：先计算出选择表达式的值，在 case 选项中找到和选择表达式值相同的第一个选择语句，然后执行此选择值后面的过程语句。

case 语句列出的选择列表，有时候不能全部包含选择表达式所有的可能值，这时关键词 default 就要被作为 case 语句的最后一个选项，它表示表达式中那些未被选择列表覆盖的所有其他值。一般情况下，即使选择列表列出了选择表达式的所有选项，还是建议保留 default 这一选项。如果选择列表中没有包含选择表达式的所有选项，而此时又没有 default 选项，那么综合器会综合出一个锁存器，以保存未被覆盖的情况下输出的过去值。这一般是不希望出现的情况，所以在 case 语句中建议无论如何保留 default 选项。

程序清单 3-2 中程序的仿真图如图 3-6 所示。

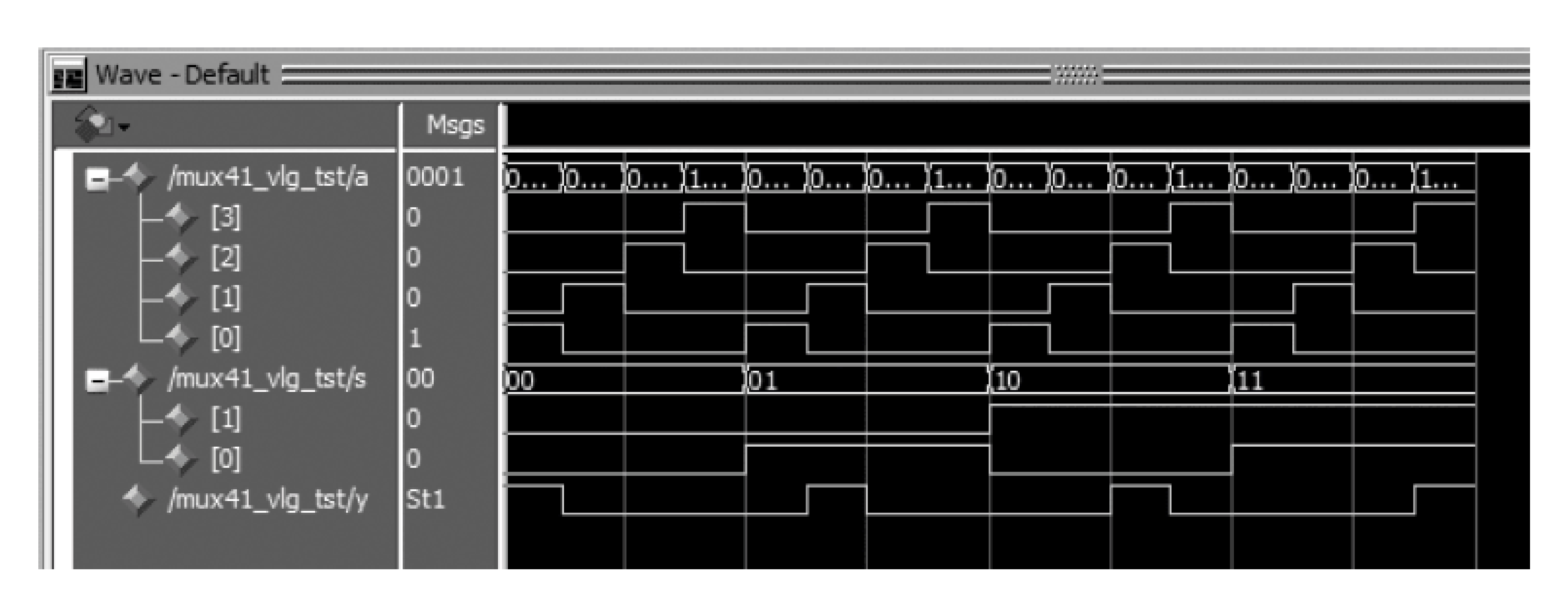


图 3-6　四选一选择器仿真图

3.1.3　实现一个多路选择器

Quartus Ⅱ 软件支持多种设计输入模型，本次实验使用 Verilog HDL 语言输入设计，在 DE2-70 开发平台上设计一个基本组合逻辑电路——二选一选择器。本实验在进一步熟悉 Quartus Ⅱ软件使用的同时，初步学习使用 Verilog HDL 语言和选择器的设计。

Verilog HDL 和 C 语言非常相像，但是两者之间存在一个重要的区别就是：C 语言中的语句是按照顺序一条一条执行的，只有上一条指令执行完了才开始下一条指令的执行。而在 Verilog HDL 中，大部分的语句、结构和各个 module 都是并发执行的。本实验的目的是着重要求 Verilog HDL 的初学者区别这一概念，根据具体事例，理解 Verilog HDL “并发执行”的概念。

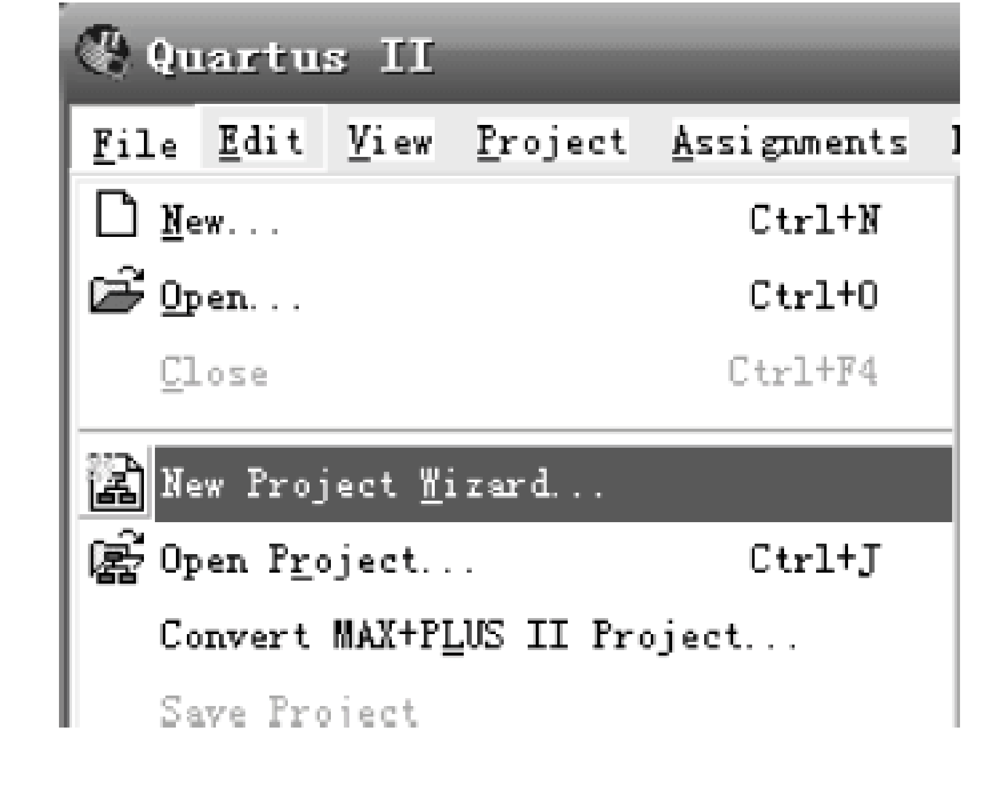


图 3-7　建立新项目

3.1.3.1　建立工程

如图 3-7 所示，建立一个工程项目。

3.1.3.2 设计输入

添加所需设计文件。本次实验通过 Verilog 语言来描述所设计的硬件，因此要添加 Verilog 语言设计文件到工程文件中去。

单击菜单项“File→New”、单击图标🗋或者使用快捷键 Ctrl+N 来新建一个设计文件，选择“Verilog HDL File”，如图 3-8 所示，单击“OK”按钮，建立 Verilog 源代码文件。

输入设计代码。在 Quartus Ⅱ 环境提供的文本编辑器中，输入用户设计的代码。Verilog 语言是一种硬件描述语言，其编写的电路是要在一定的硬件上实现的，因此想要完成一个高效的电路设计，要尽量利用编译器更能理解的方式来编写程序，也就是用符合 Verilog 语言的代码风格来编写程序。对于初学者，Quartus Ⅱ 环境给我们提供了常见的代码模板，从模块的结构、简单寄存器的编写直到一些简单电路的完整设计都有。

在 Verilog 文本编辑器的空白处单击鼠标右键，选择“Insert Template”选项，如图 3-9 所示。

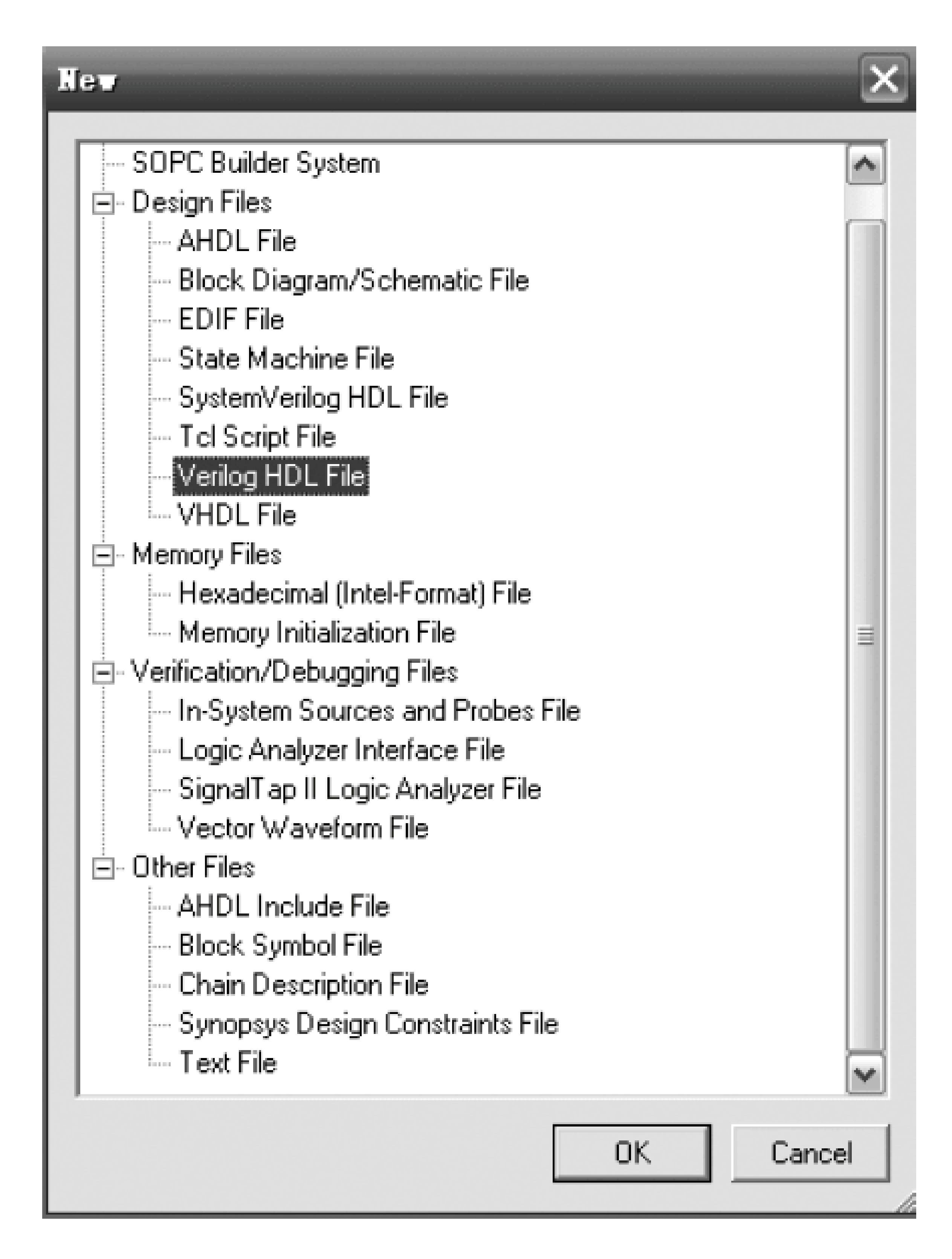


图 3-8 新建 Verilog 语言文件

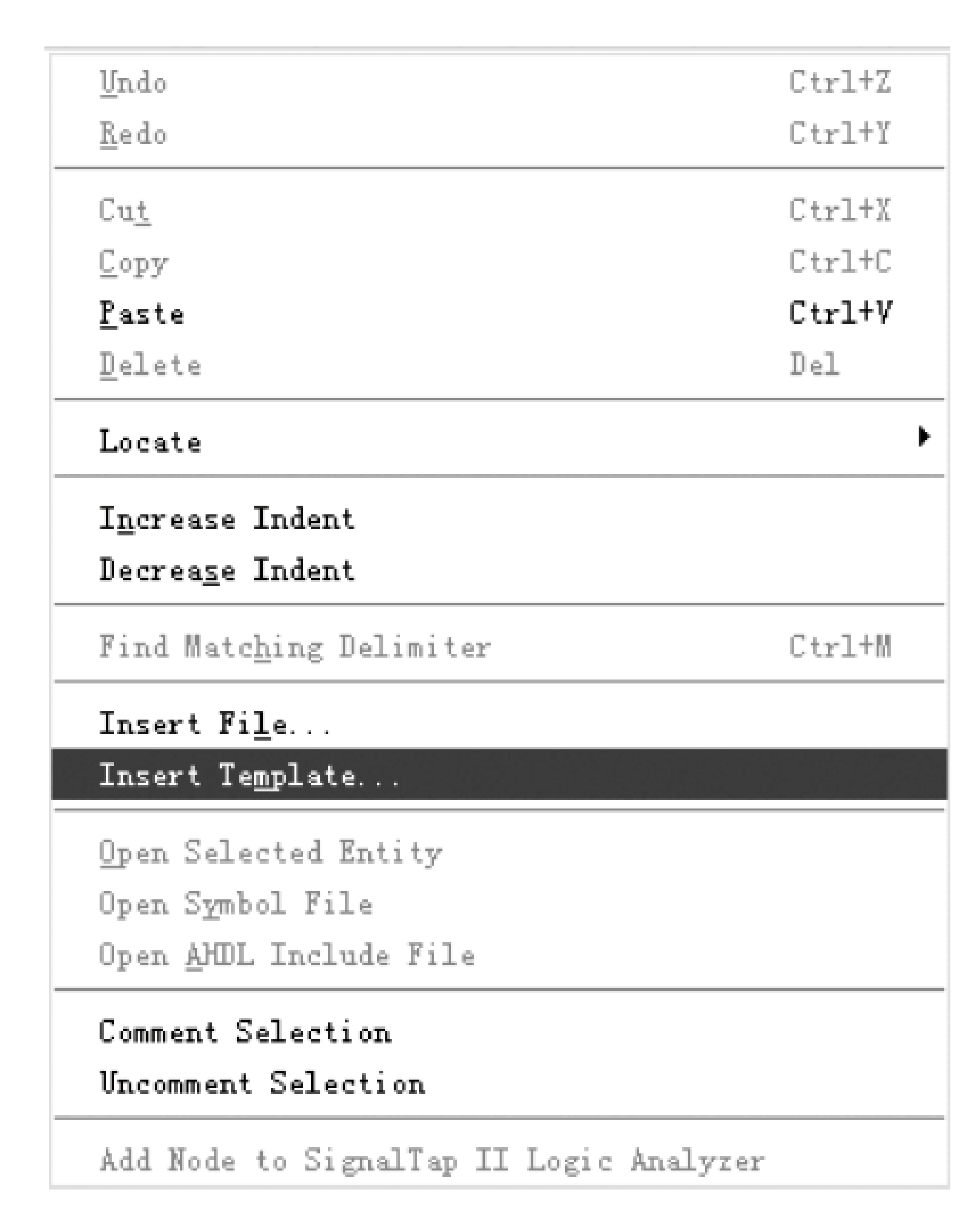


图 3-9 利用 Verilog 语言模板

在弹出的对话框左侧的树状目录中，展开 Verilog HDL，可以寻找到一些常用的模板，如图 3-10 所示。

选择好要加入的模板后，单击“Insert”按钮即可加入该模板。

本例实验设计的是一个二选一的选择器，我们来验证上一节所述的门级描述语言的方式，代码很简单，直接输入，输入的代码如图 3-11 所示。从代码中可以看出，为了方便，下面要使用 DE2-70 开发板上的输入输出器件来验证我们的设计，在变量取名上我们按照输入输出器件的名称来取名。

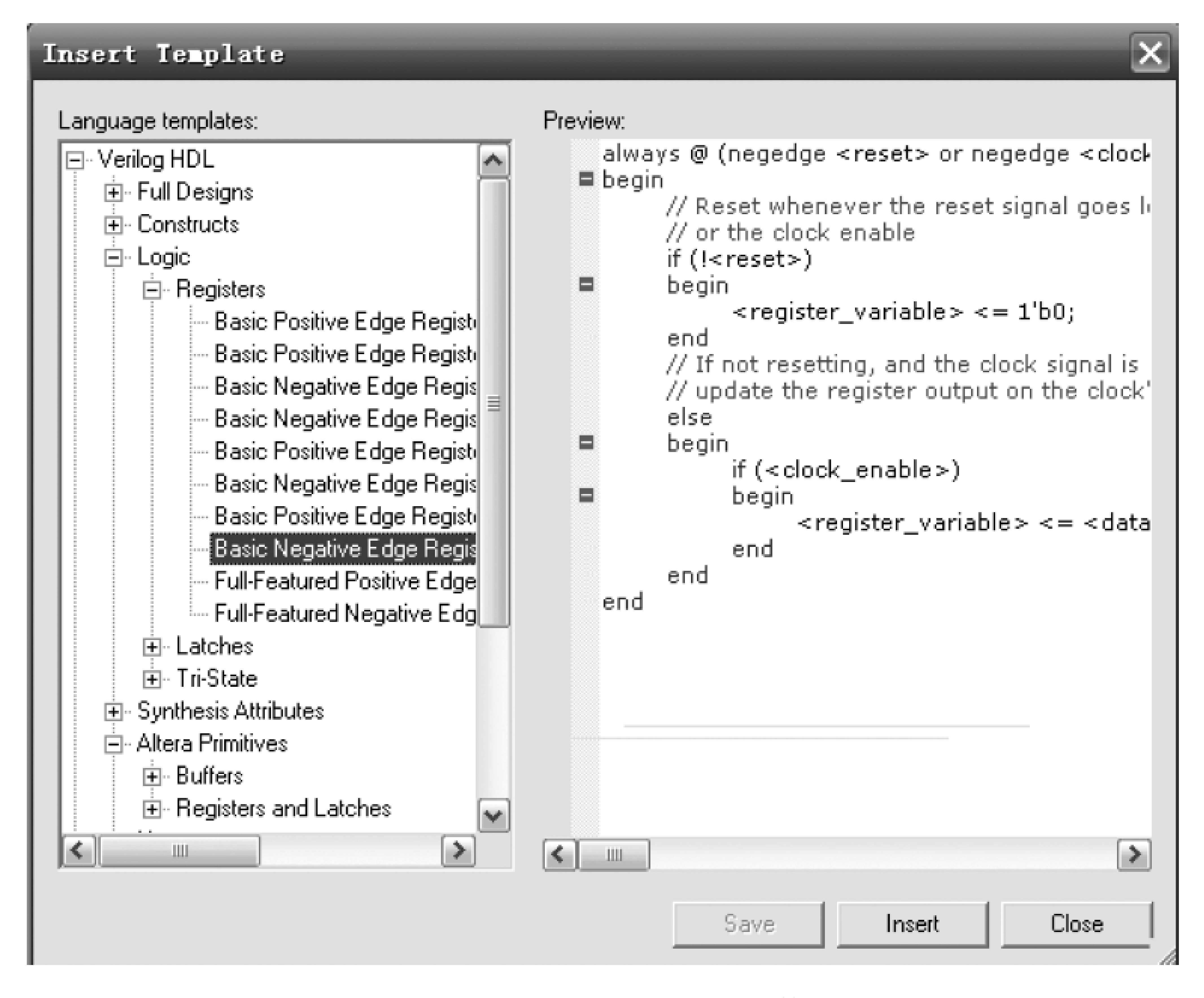


图 3-10　选择 Verilog 语言模板

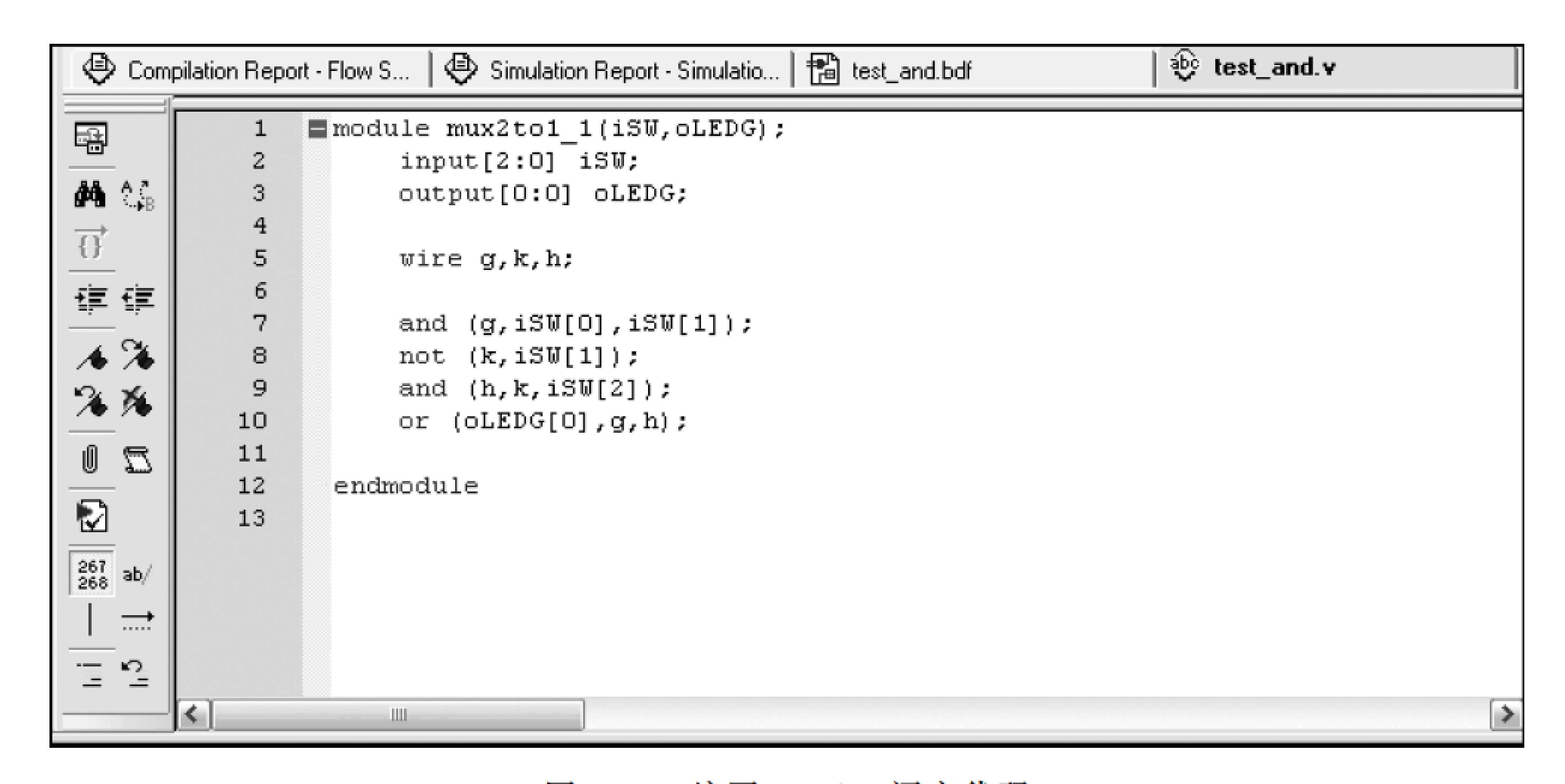


图 3-11　编写 Verilog 语言代码

3.1.3.3　分析与综合

输入代码后对设计好的程序进行分析与综合，检查并修改代码。

3.1.3.4　建立仿真测试文件

分析与综合通过后，建立仿真测试文件，对综合出的电路进行功能仿真，如图 3-12、图 3-13 和图 3-14 所示。

```
`timescale 10 ns/ 1 ps
module mux2to1_1_vlg_tst();
// constants
// test vector input registers
reg [2:0] iSW;
// wires
wire [0:0]  oLEDG;

// assign statements (if any)
mux2to1_1 i1 (
// port map - connection between master ports an
    .iSW(iSW),
    .oLEDG(oLEDG)
);
initial
begin
// code executes for every event on sensitivity
// insert code here --> begin
iSW[2]=0; iSW[1]=0; iSW[0]=0; #10;
iSW[2]=0; iSW[1]=0; iSW[0]=1; #10;
iSW[2]=0; iSW[1]=1; iSW[0]=0; #10;
iSW[2]=0; iSW[1]=1; iSW[0]=1; #10;
iSW[2]=1; iSW[1]=0; iSW[0]=0; #10;
iSW[2]=1; iSW[1]=0; iSW[0]=1; #10;
iSW[2]=1; iSW[1]=1; iSW[0]=0; #10;
iSW[2]=1; iSW[1]=1; iSW[0]=1; #10;
// --> end
end
endmodule
```

图 3-12　修改测试文件

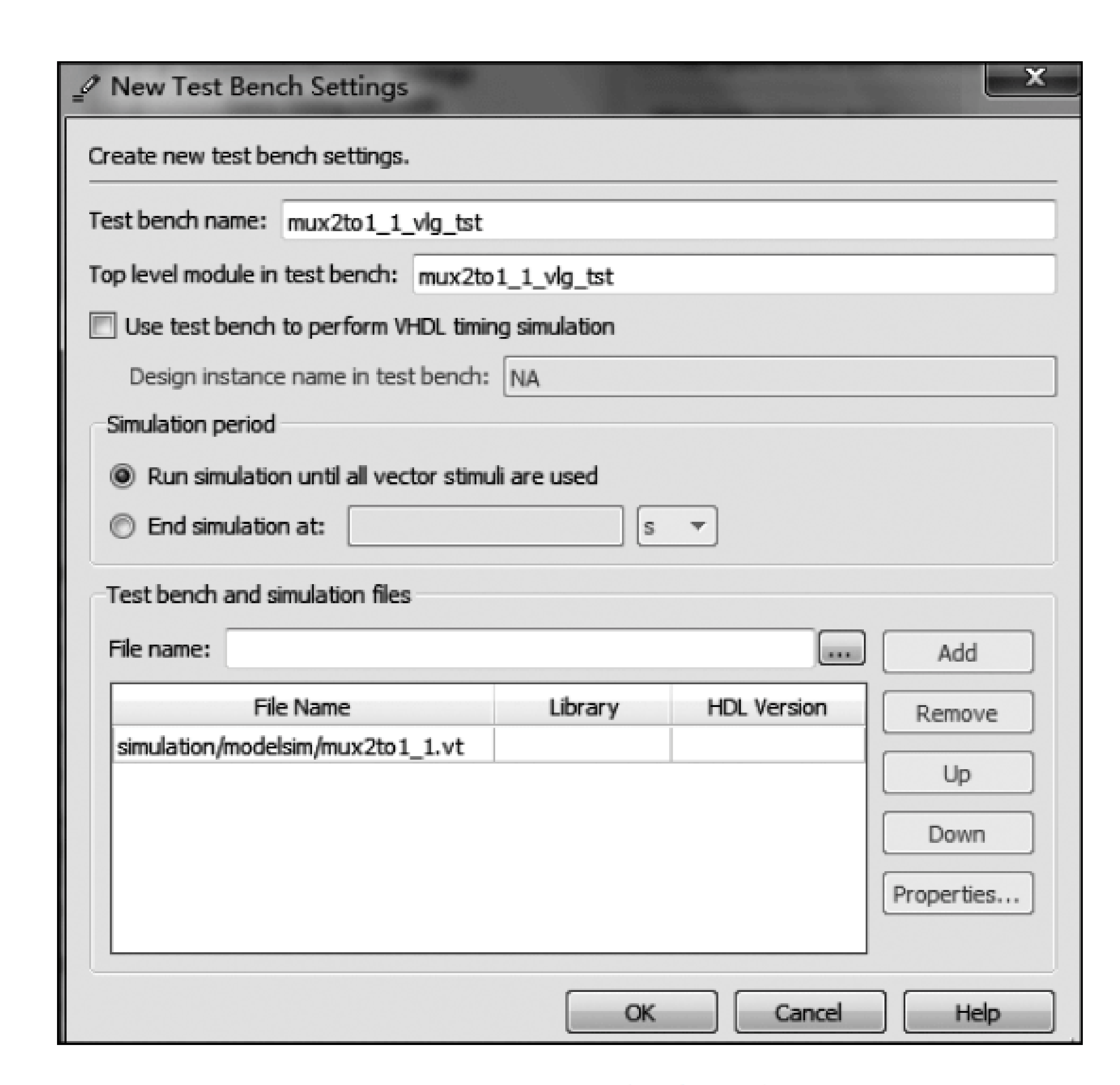


图 3-13　设置仿真文件

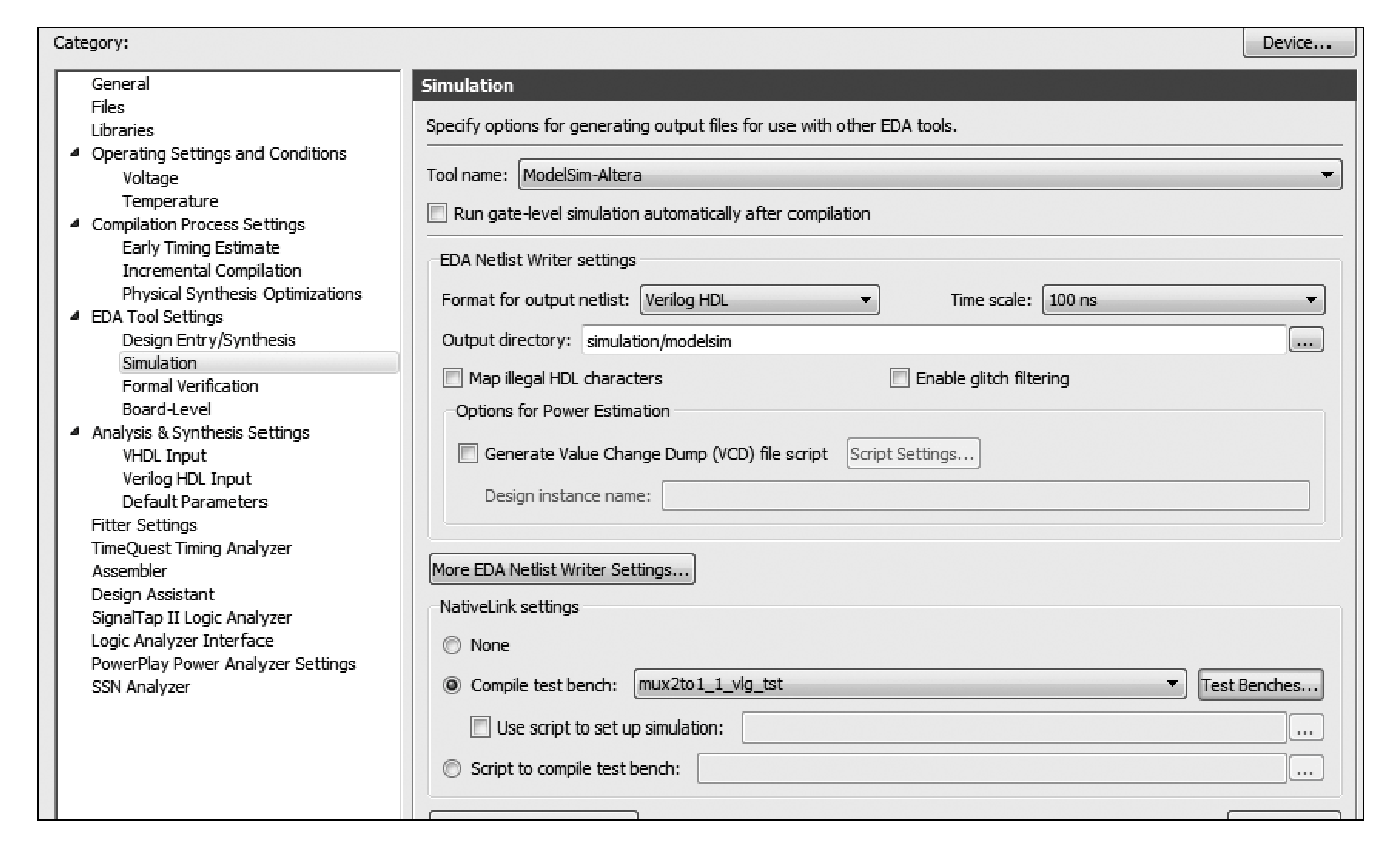


图 3-14　选择仿真选项

3.1.3.5　分配引脚

本书在第 2 章中，曾经介绍过采用手工分配的方式来给电路分配引脚，本实验中，将采用另外一种导入引脚文件的方式来分配引脚，两种分配引脚的方式是等效的。

用导入引脚配置文件的方式分配引脚，是编译器自动地根据引脚配置列表将 FPGA 上的引脚全部分配，这样分配引脚的前提是设计者在设计电路的时候，输入输出引脚取的名字应该和配置文件中的引脚名一样，否则无法自动分配引脚。选择"Assignment→Import Assignment"选项，如图 3-15 所示。

弹出输入引脚配置文件路径对话框，如图 3-16 所示。

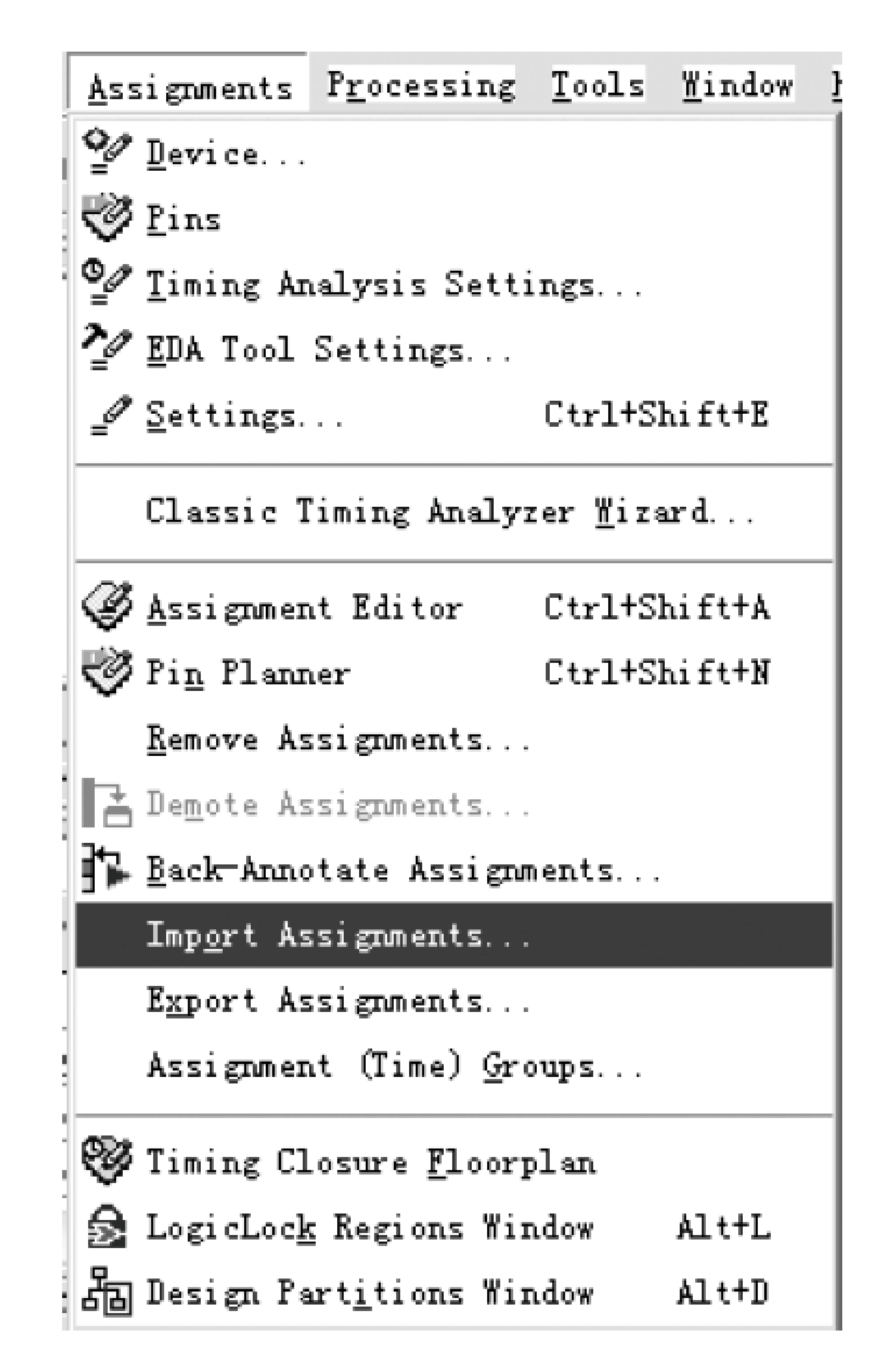


图 3-15 导入引脚文件

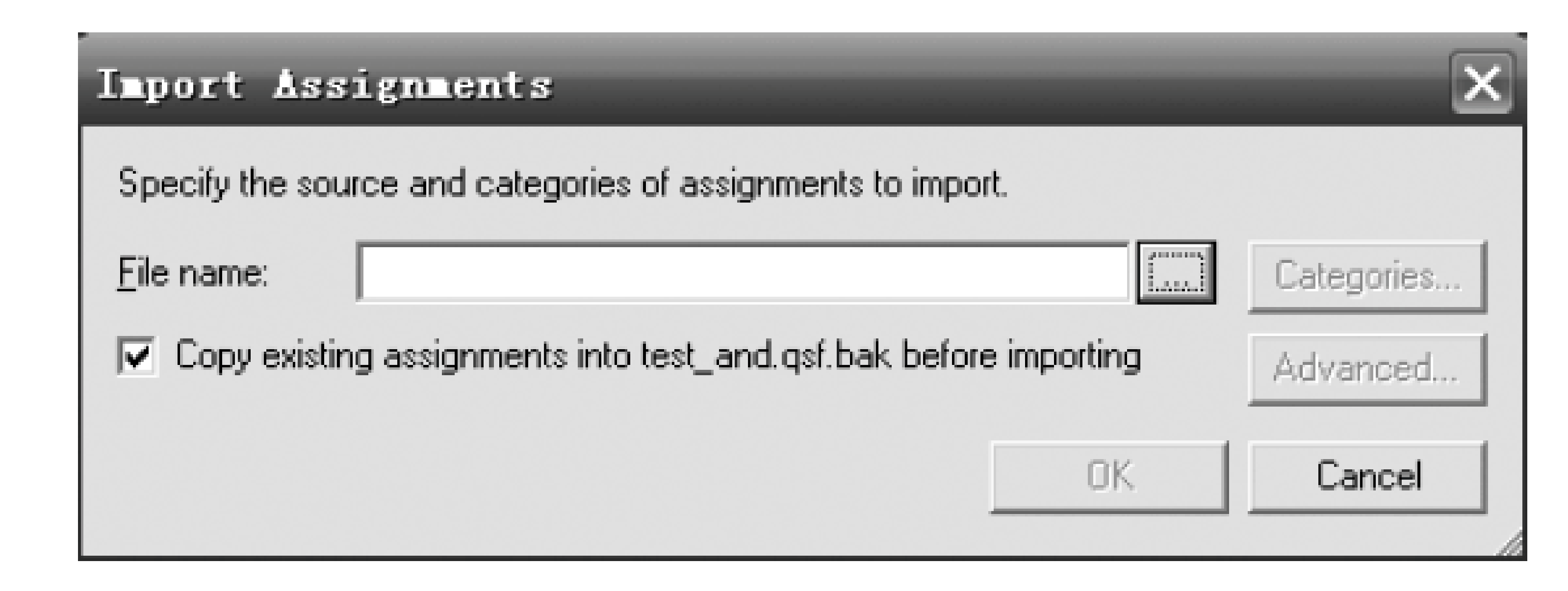


图 3-16 选择引脚配置文件路径

引脚分配文件就是前面所讲述的在开发板配套的光盘 DE2-70 System CD-ROM 中的“DE2_70_pin_assignments. csv”文件，选择正确路径，将其导入。

导入引脚分配文件后查看引脚分配面板会发现，所有的 FPGA 的外接引脚全部被导入，如图 3-17 所示。

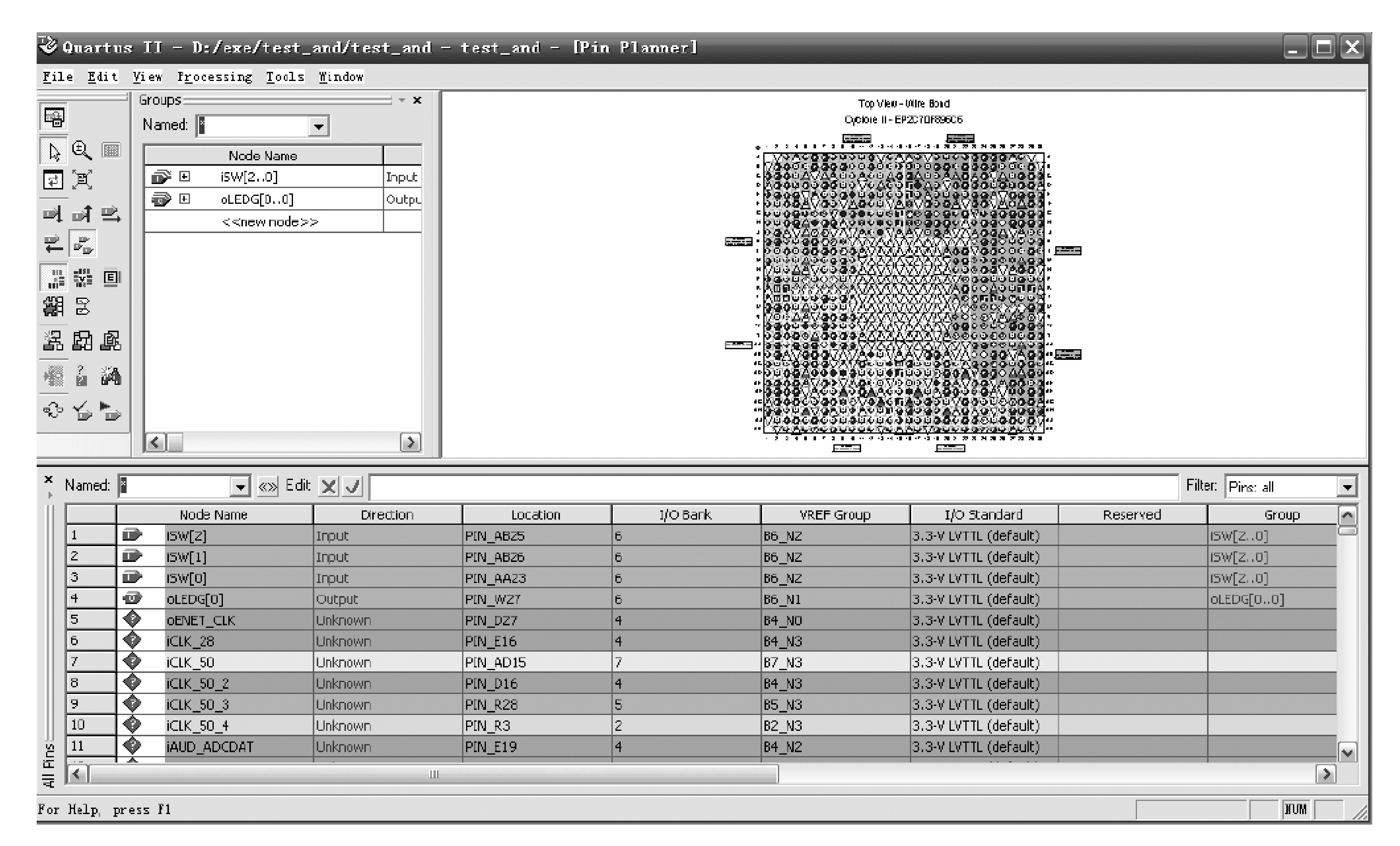


图 3-17 导入引脚后引脚配置窗口

导入全部的引脚后，进行全编译会出现大量的警告，这主要是因为引脚分配文件中含有大量的引脚在本次实验中没有被用到，一般情况下，可忽略这些警告，如图 3-18 和图 3-19 所示。



图 3-18　全编译成功

```
Type  Message
Warning: Ignored locations or region assignments to the following nodes
   Warning: Node "AUD_ADCLRCK" is assigned to location or region, but does not exist in design
   Warning: Node "AUD_BCLK" is assigned to location or region, but does not exist in design
   Warning: Node "AUD_DACLRCK" is assigned to location or region, but does not exist in design
   Warning: Node "DRAM_DQ[0]" is assigned to location or region, but does not exist in design
   Warning: Node "DRAM_DQ[10]" is assigned to location or region, but does not exist in design
   Warning: Node "DRAM_DQ[11]" is assigned to location or region, but does not exist in design
   Warning: Node "DRAM_DQ[12]" is assigned to location or region, but does not exist in design
   Warning: Node "DRAM_DQ[13]" is assigned to location or region, but does not exist in design
   Warning: Node "DRAM_DQ[14]" is assigned to location or region, but does not exist in design
   Warning: Node "DRAM_DQ[15]" is assigned to location or region, but does not exist in design
   Warning: Node "DRAM_DQ[16]" is assigned to location or region, but does not exist in design
   Warning: Node "DRAM_DQ[17]" is assigned to location or region, but does not exist in design
   Warning: Node "DRAM_DQ[18]" is assigned to location or region, but does not exist in design
   Warning: Node "DRAM_DQ[19]" is assigned to location or region, but does not exist in design
System (20) | Processing (52) | Extra Info | Info (49) | Warning (3) | Critical Warning | Error | Suppressed (6) | Flag
```

图 3-19　查看 Warning

最好是删除这些没有使用的引脚：用记事本等外部编辑器打开本工程目录下的“.qsf”文件，本例中是“mux2to1_1.qsf”。发现大量的“set_location_assignment PIN_XXX -to YYYY”信息。将不用的引脚配置信息删除，留下需要使用的几个引脚：

```
set_location_assignment PIN_AA23- to iSW[0]
set_location_assignment PIN_AB26- to iSW[1]
set_location_assignment PIN_AB25- to iSW[2]
set_location_assignment PIN_W27- to oLEDG[0]
```

回到 Quartus Ⅱ 中重新观察引脚会发现，只留下了本工程中使用到的几个引脚，如图 3-20 所示。

请将其他未使用的管脚设置为三态，双击 Quartus Ⅱ 左上角的 Cyclone Ⅱ：EP2C70F896C6。然后，单击“device and Pins Options”选项，选择“Unused Pins”栏，将其设置为“As Input tri-stated”。

3.1.3.6　重新全编译

重新全编译后，那些关于未使用的引脚的大量警告信息已经消除，如图 3-21 所示。

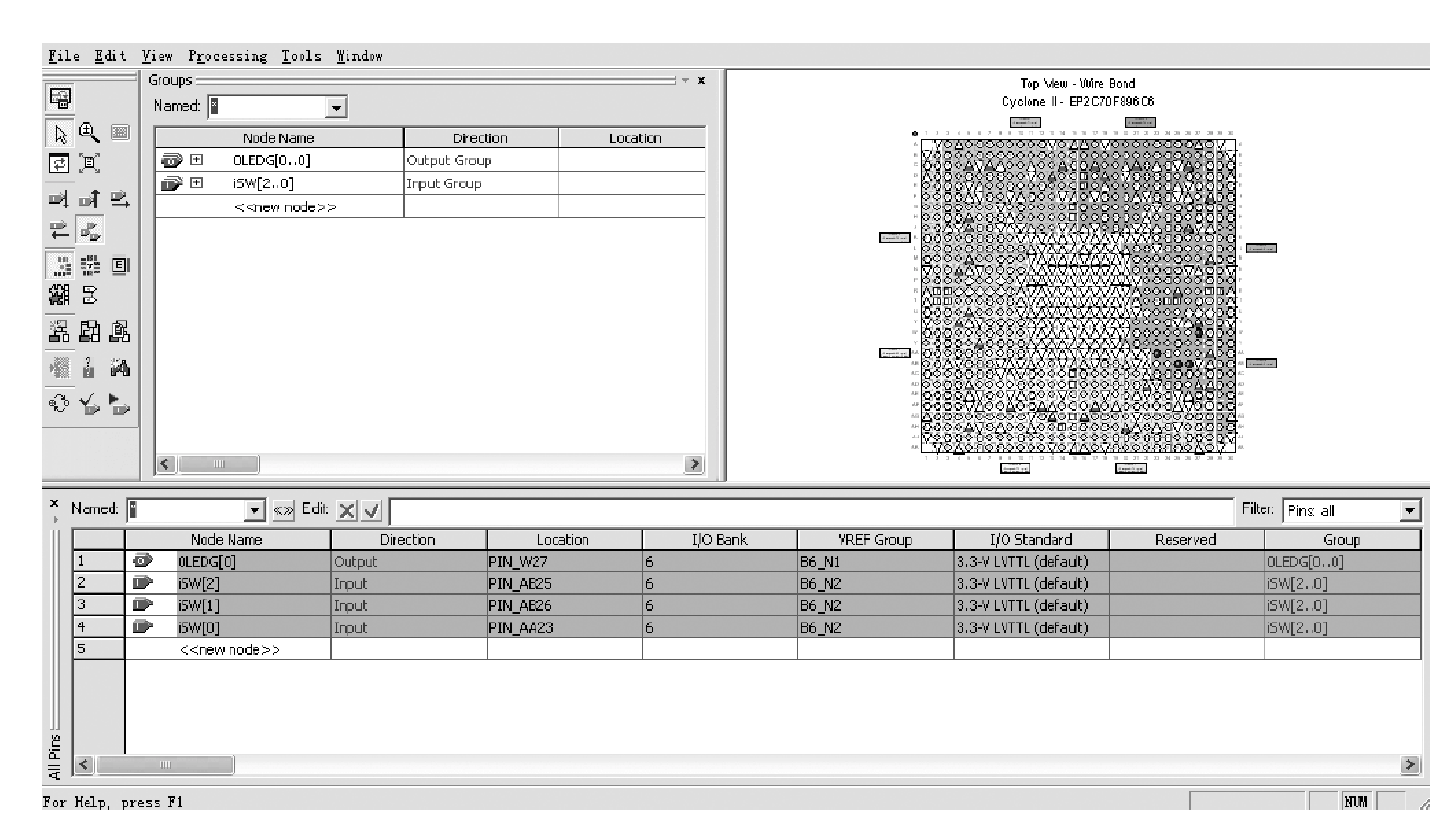


图 3-20　引脚配置详情



图 3-21　全编译结果

3.1.3.7　时序仿真

全编译后需要对电路进行时序仿真，如图 3-22 所示。



图 3-22　时序仿真结果

3.1.3.8　完成下载

将设计下载到 FPGA 中，分析时序仿真结果正确后，就可以将电路下载至开发板进行验证了，如图 3-23 所示。下载完成后拨动开发板上的开关，检查设计结果是否满足要求。

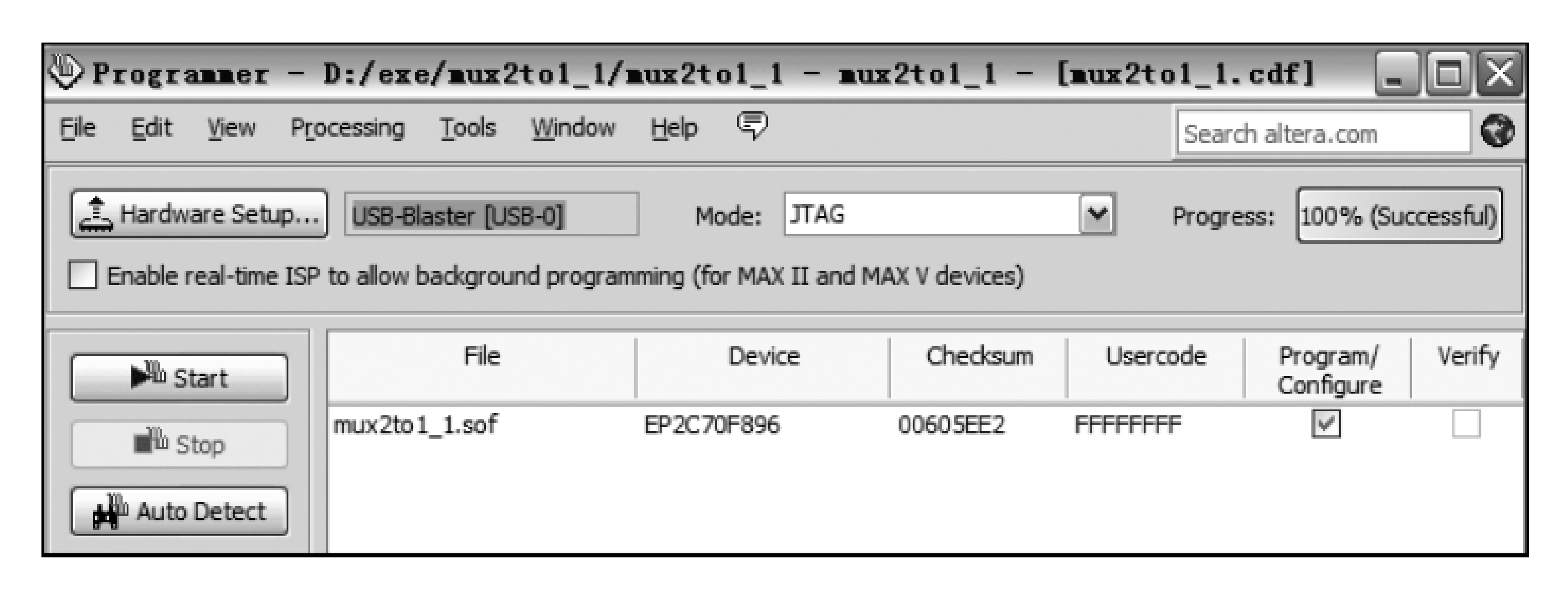


图 3-23　下载完成

3.1.4　实验内容

3.1.4.1　8 位二选一选择器

用 if 语句实现一个 8 位二选一的选择器，如图 3-24 所示。选择器有两个输入端，分别为 X 和 Y，输出端为 F。输出端受控制端的控制，当控制端为 0 时，输出端输出 X，即 $F=X$；当控制端为 1 时，输出端输出 Y，即 $F=Y$。

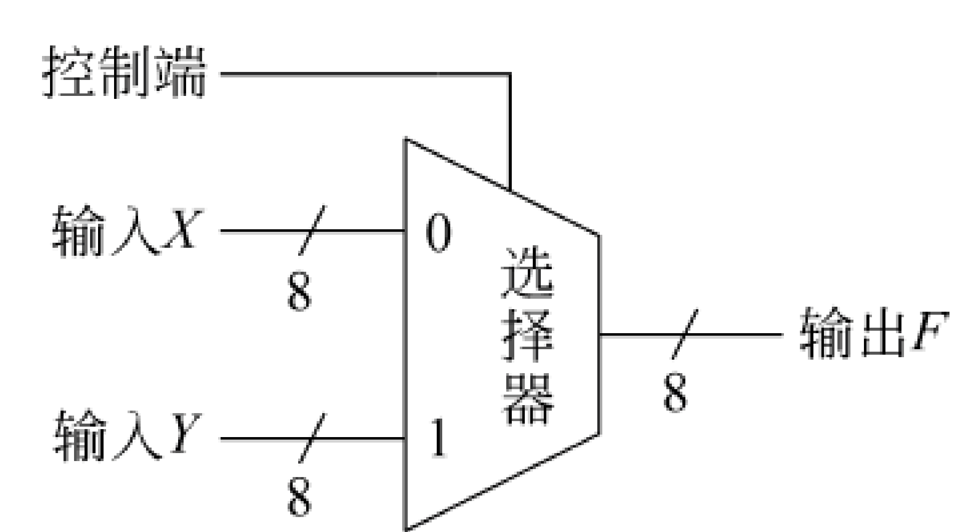


图 3-24　8 位二选一选择器

选择 DE2-70 板上的 iSW[8]作为控制端，iSW[0]～iSW[7]作为输入端 X，iSW[10]～iSW[17]作为输入端 Y，将输出端 F 接到数码管或者发光二极管上显示输出，完成设计并下载到开发板上验证电路性能。

提示：本实验中，在导入引脚配置文件后，进行全编译时，会出现一个错误信息，这是 DE2-70 开发板的设置问题，如图 3-25 所示。

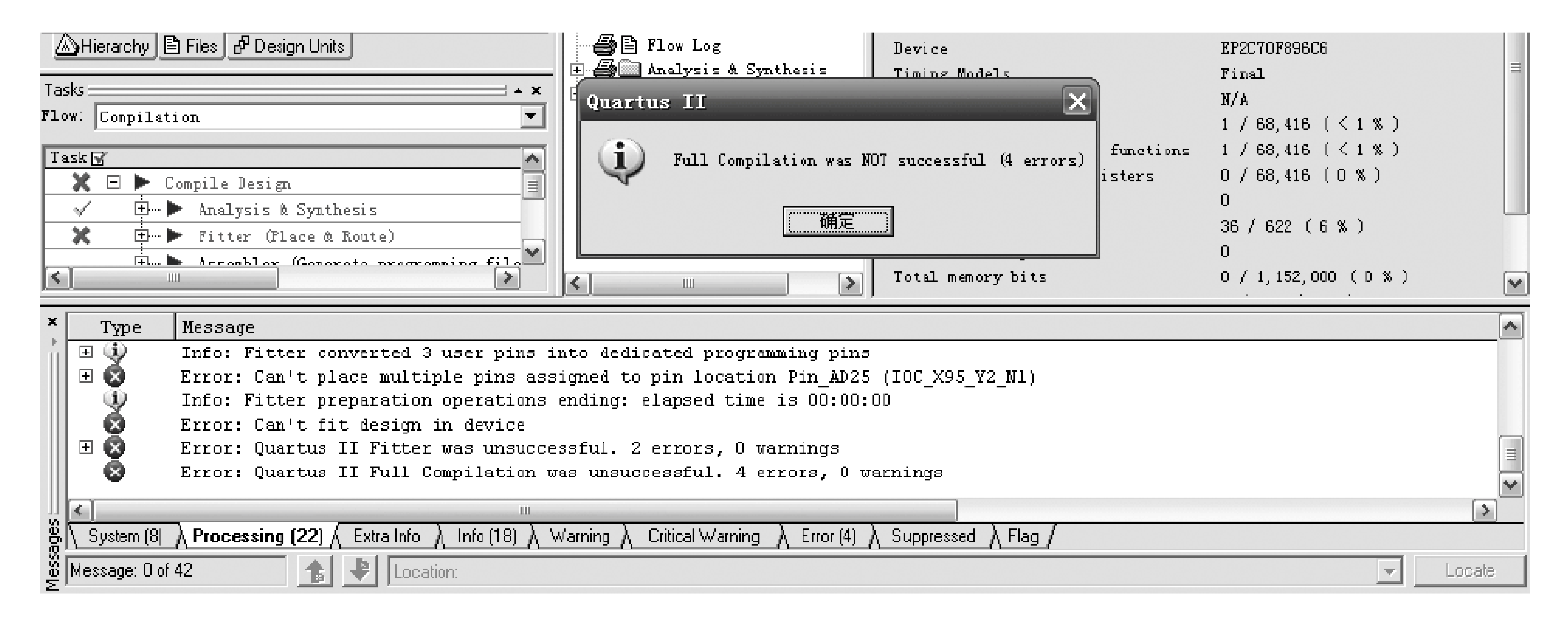


图 3-25　引脚配置报错

解决方法是：单击“Assignment→Device”选项，弹出 Device 设置对话框，单击“Device and Pin Options”选项，如图 3-26 所示。

在弹出的对话框中选择“Dual-Purpose Pins”选项，双击“Use as programming pin”选项，在弹出的下拉菜单中选择“Use as regular I/O”选项，如图 3-27 所示。确定、重新编译即可。

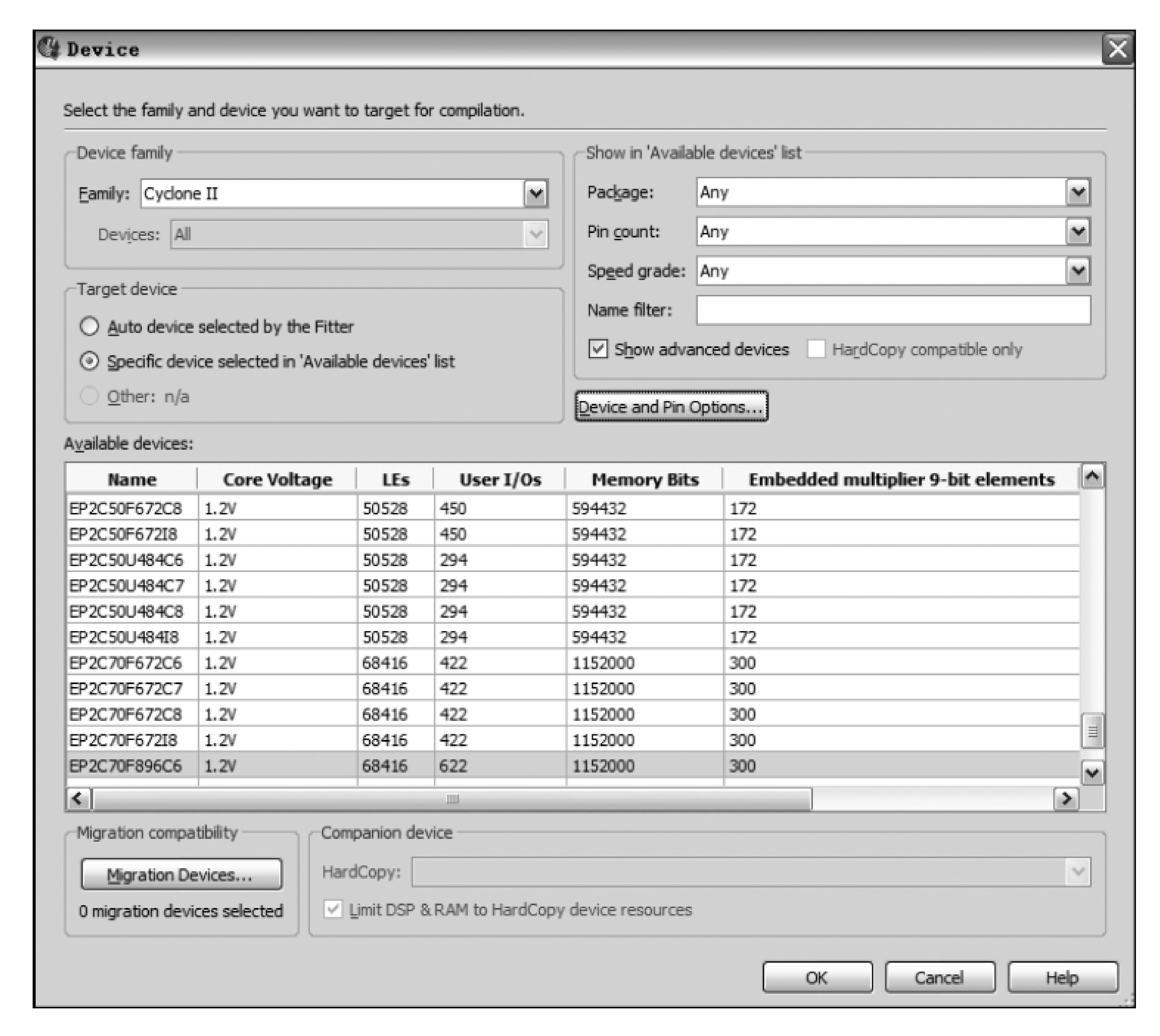


图 3-26　选择引脚配置项

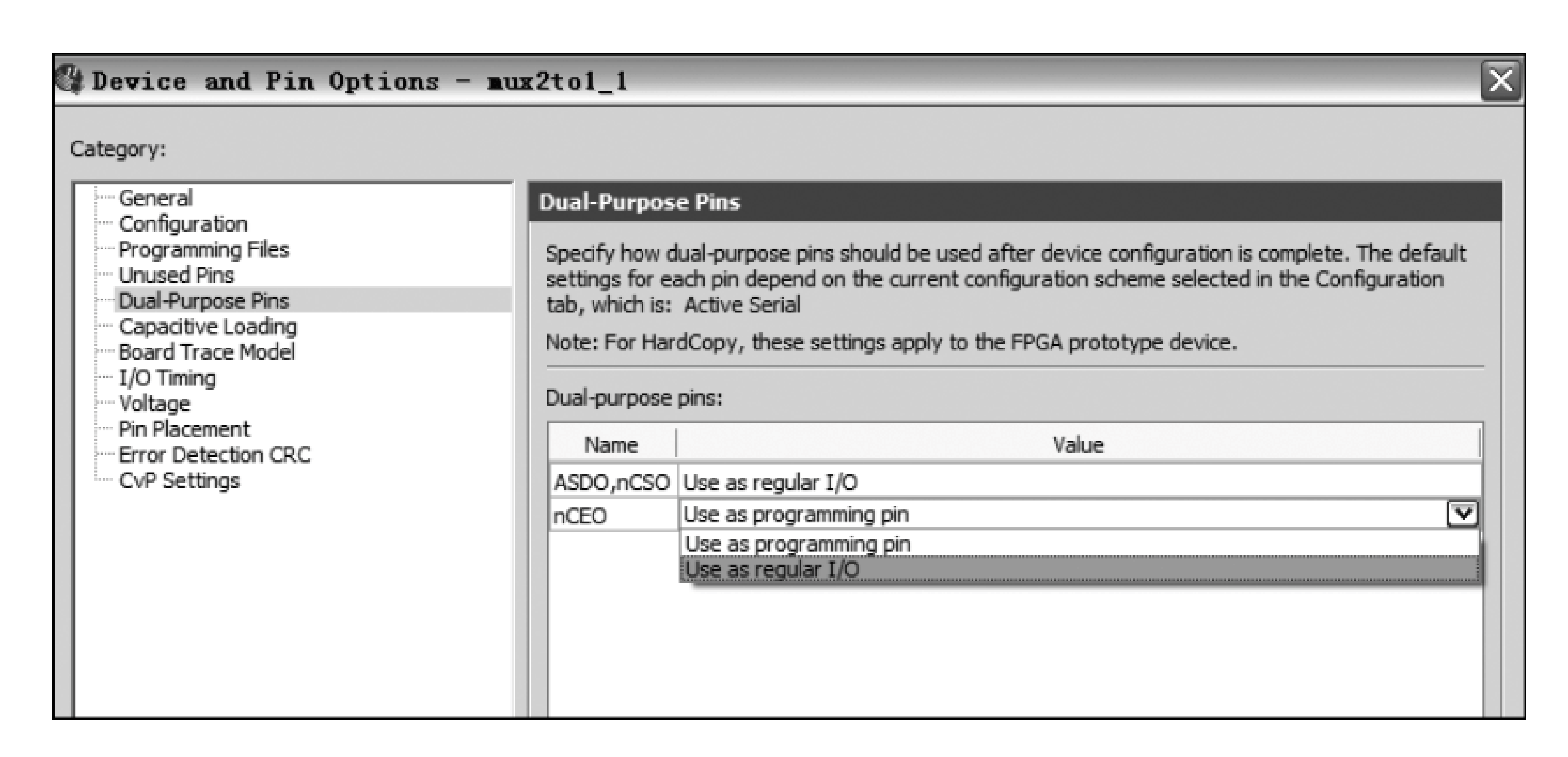


图 3-27　修改引脚配置

3.1.4.2　3 位五选一的选择器

用 case 语句实现一个 3 位五选一的选择器，如图 3-28 所示。$Z_0 \sim Z_2$ 是控制端，5 个输入端 $X_0 \sim X_4$，输出端是 Y。

请选择 DE2-70 板上的适当端口作为控制端、输入端和输出端，将电路下载至 DE2-70 开发板上，验证其功能。

提示：本实验中，在导入引脚配置文件后，进行全编译时，会出现一个错误信息，这是 DE2-70 开发板的设置问题，如图 3-29 所示。

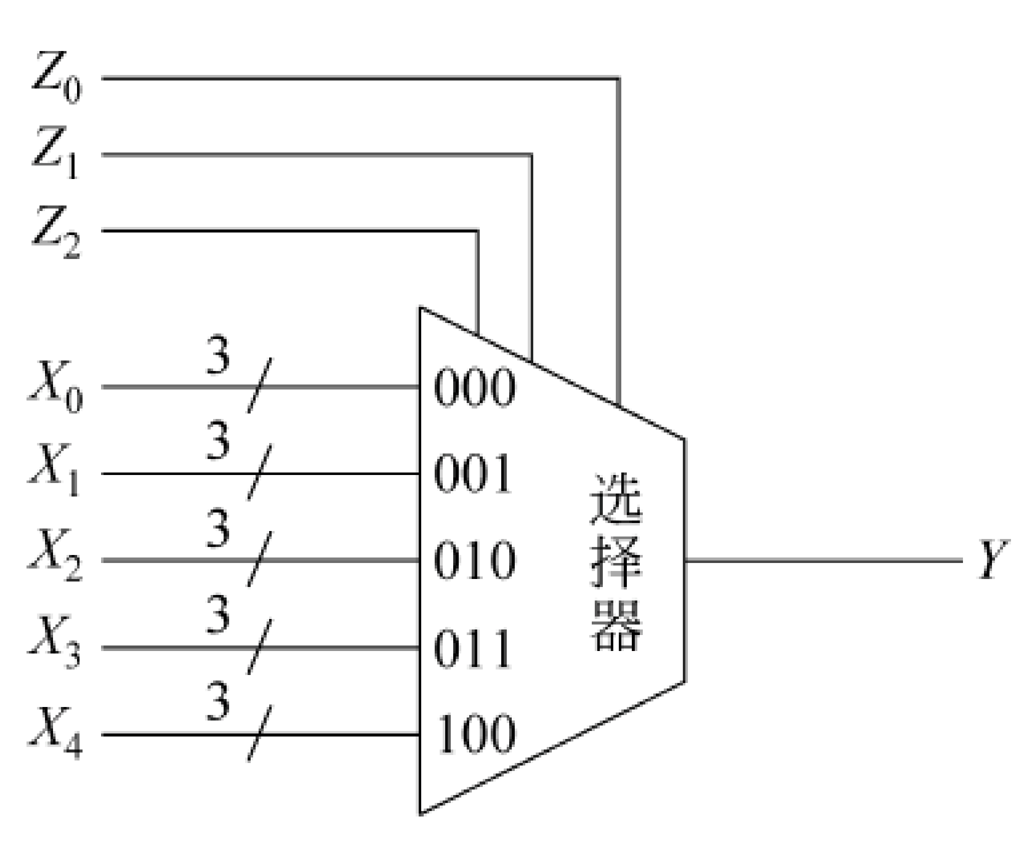


图 3-28　3 位五选一选择器

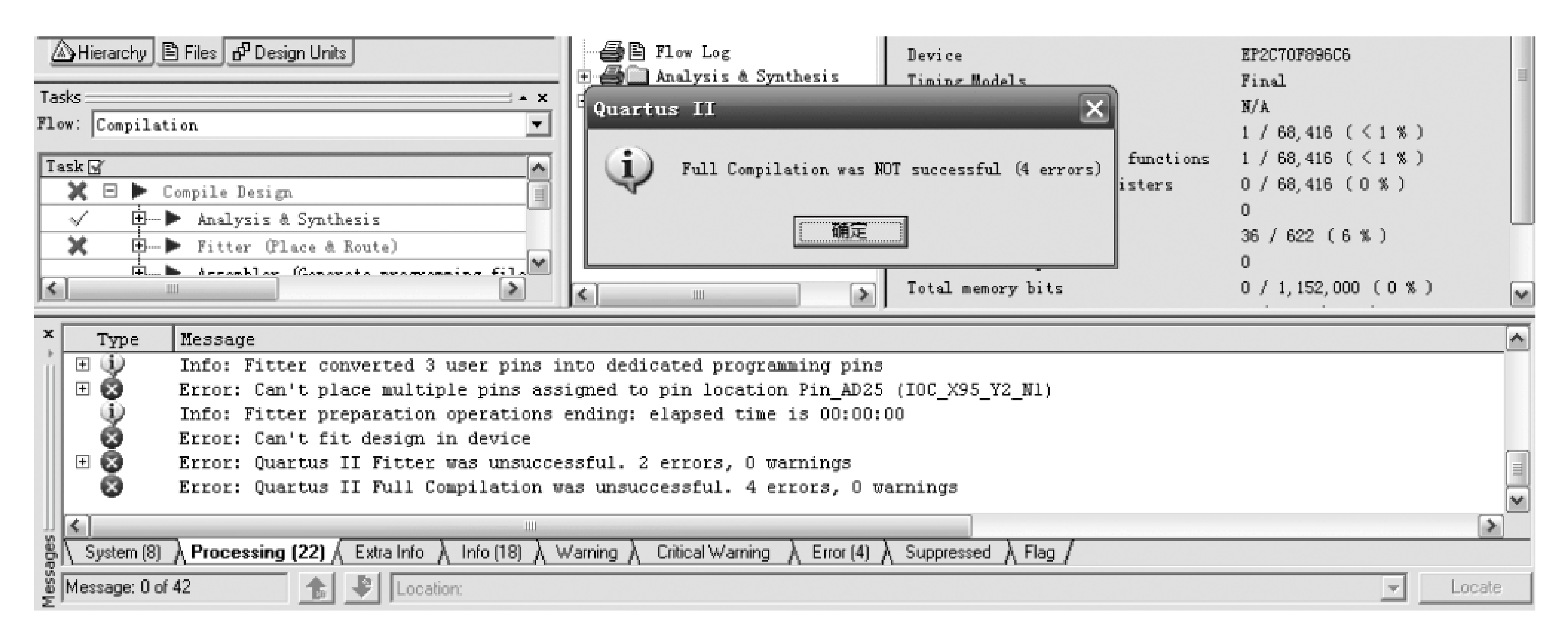


图 3-29　引脚配置报错

解决方法是：单击“Assignment→Device”选项，弹出 Device 设置对话框，单击“Device and Pin Options”选项，如图 3-30 所示。

在弹出的对话框中选择“Dual-Purpose Pins”选项，双击“Use as programming pin”选项，在弹出的下拉菜单中选择“Use as regular I/O”选项，如图 3-31 所示。确定后重新编译即可。

3.1.4.3* 桶形移位器

桶形移位器(barrel shifter)是一种组合逻辑电路，它输入一个 n 位的数据，输出一个 n 位的数据。其工作原理是：根据控制信号，将输入数据在一次操作中移动数位，然后从输出端输出。由于桶形移位器一次可以移动数位数据，不需要时钟控制，工作速度非常快，已经成为 CPU 重要的一部分。

图 3-32 是一个由若干个四选一的选择器构成的 4 位桶形移位器逻辑图，其输入端是 $D_0 \sim D_3$，输出端是 $Q_0 \sim Q_3$，两个移位方式控制端口 C_0 和 C_1，控制端控制移位器的工作方式，能够完成算术/逻辑的左移/右移 4 个功能，由 S_0 和 S_1 决定桶形移位器一次移动几位。

本项实验要求完成此 4 位的桶形移位器，请仔细分析其工作原理，体会桶形移位器工作的过程。请预先设计一个四选一的选择器，再利用此选择器完成桶形移位器的设计，设计过程可参考实验 3.3。

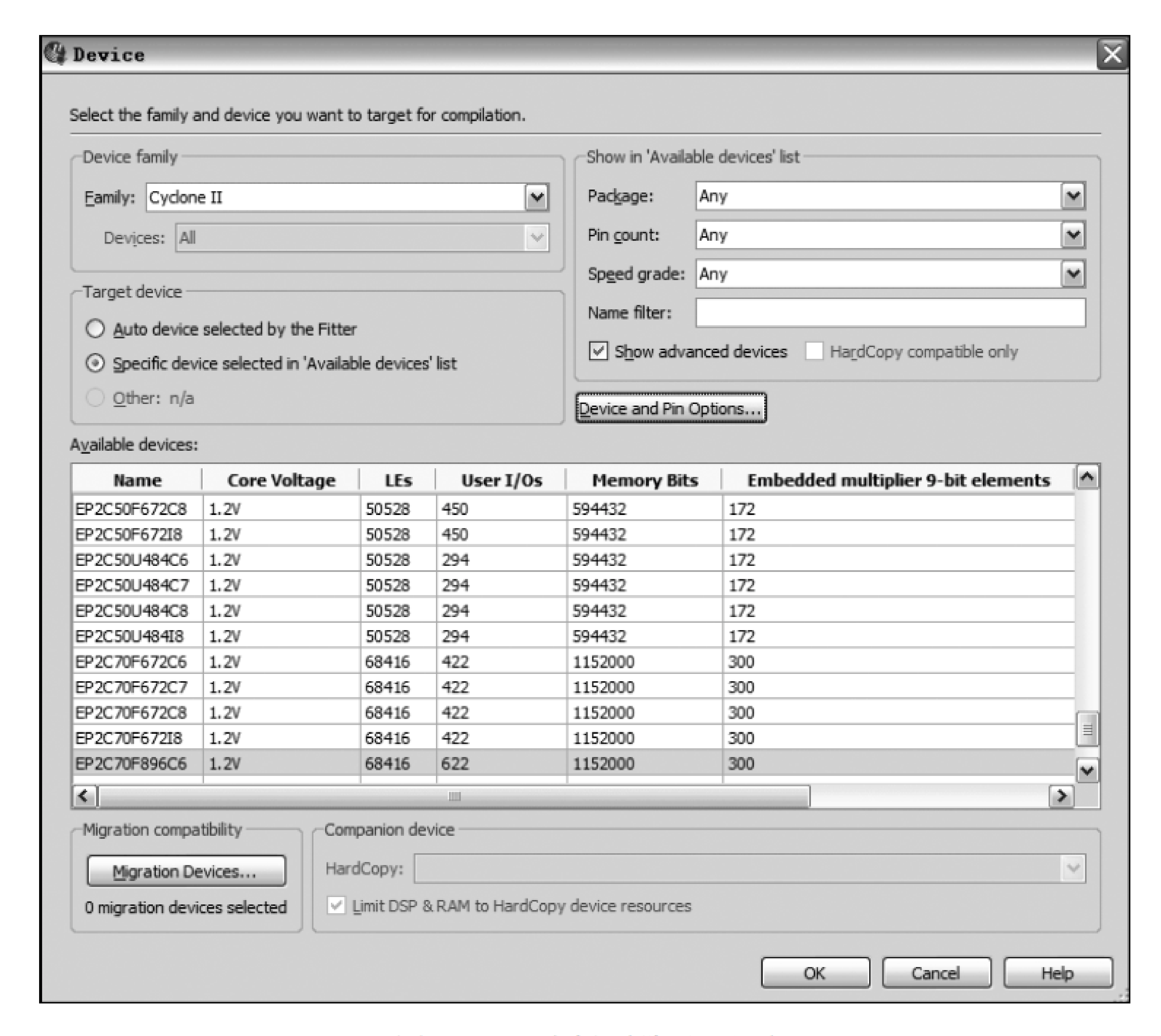


图 3-30 选择引脚配置项

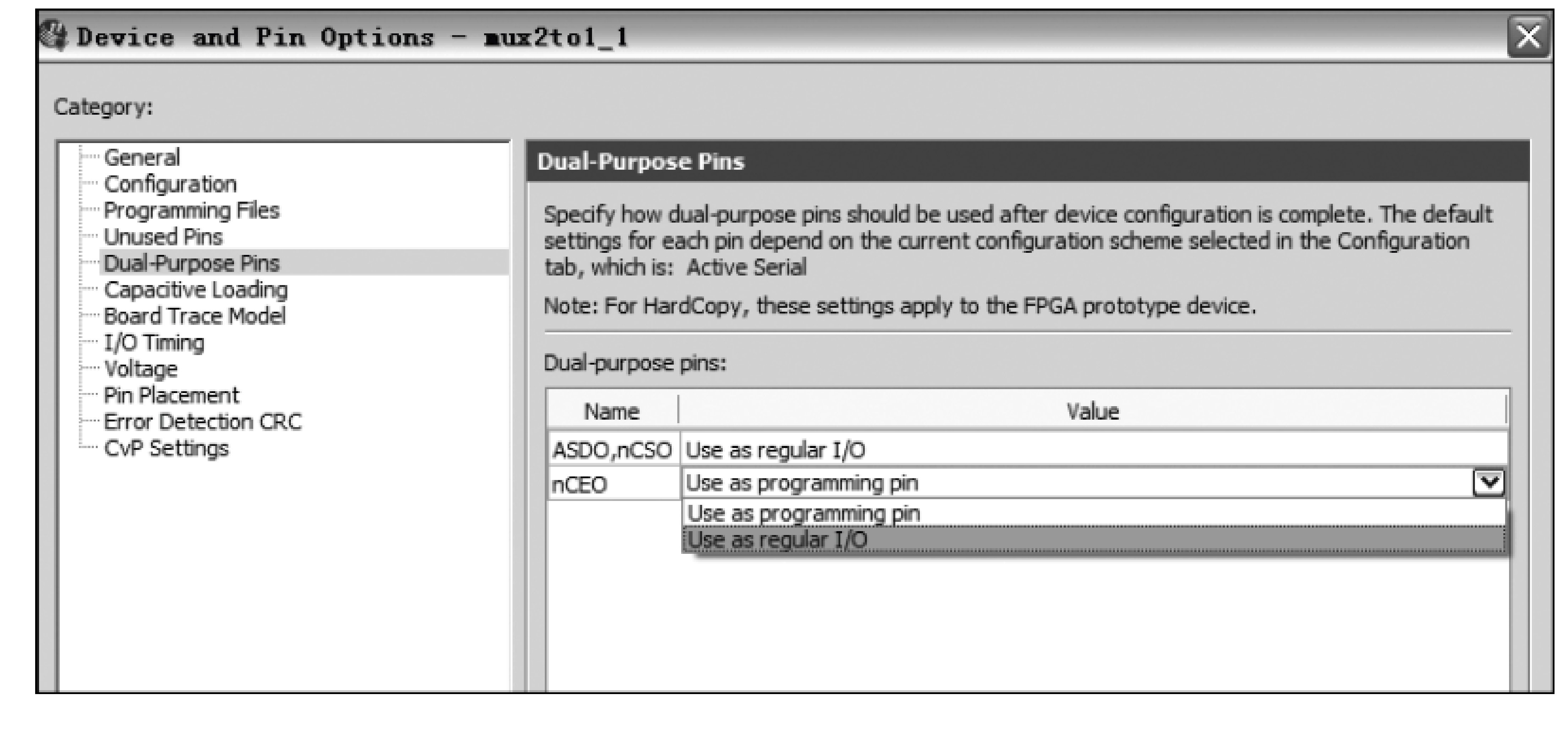


图 3-31 修改引脚配置



图 3-32　4 位桶形移位器

3.2 译码器的设计

译码器也是组合逻辑电路的一个重要器件，译码器是将某一个输入信息转换为某一个特定输出的逻辑电路，通常是多路输入输出电路，它将 n 位的输入编码转换为 m 位的编码输出，一般情况下 $n<m$，输入编码和输出编码之间存在着一一对应的映射关系，每个输入编码产生唯一的一个不同于其他的输出编码。

常用的二进制译码器是一个有 n 路输入和 2^n 路输出的逻辑电路，如图 3-33 所示。在 2^n 路输出信号中，只有一个输出信号有效。译码器有一个使能信号 E_n，当 $E_n=0$ 时，无论输入是什么，译码器都没有任何有效值输出；当 $E_n=1$ 时，输入的值决定了输出信号的值。在二进制码中，最常用的输出编码是 m 位中取 1 位编码(设输出为 m 位码)，即任何时刻，m 位输出编码中只能有 1 位有效，其余各位都为 0，这样的二进制编码被称为独热编码(one-hot encoded)，意思是那个被置为“1”的码看起来是“热”的，而二进制译码器输出的信号就是独热编码。

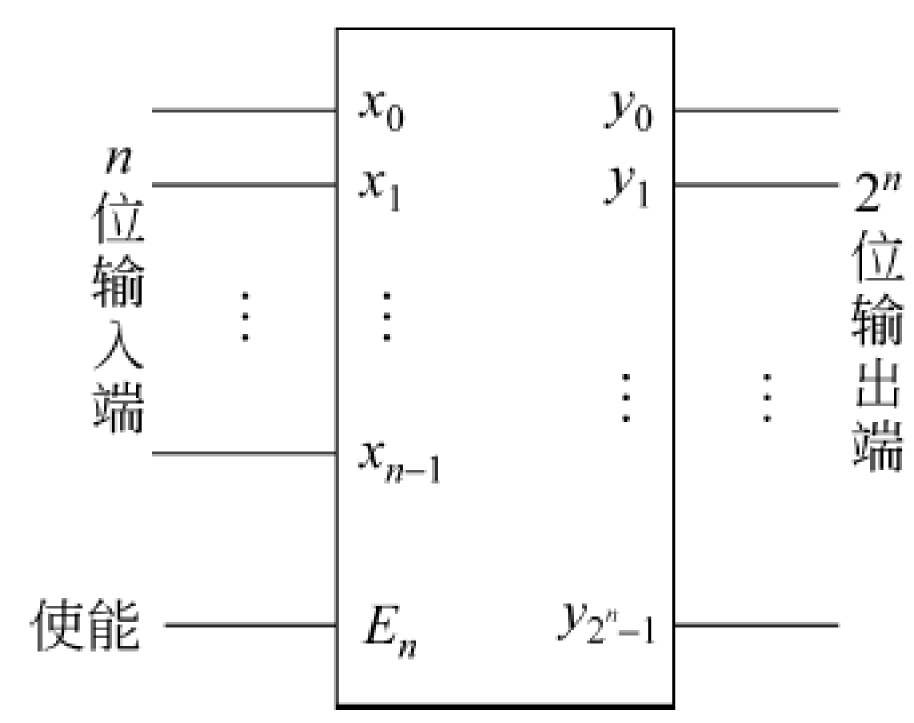


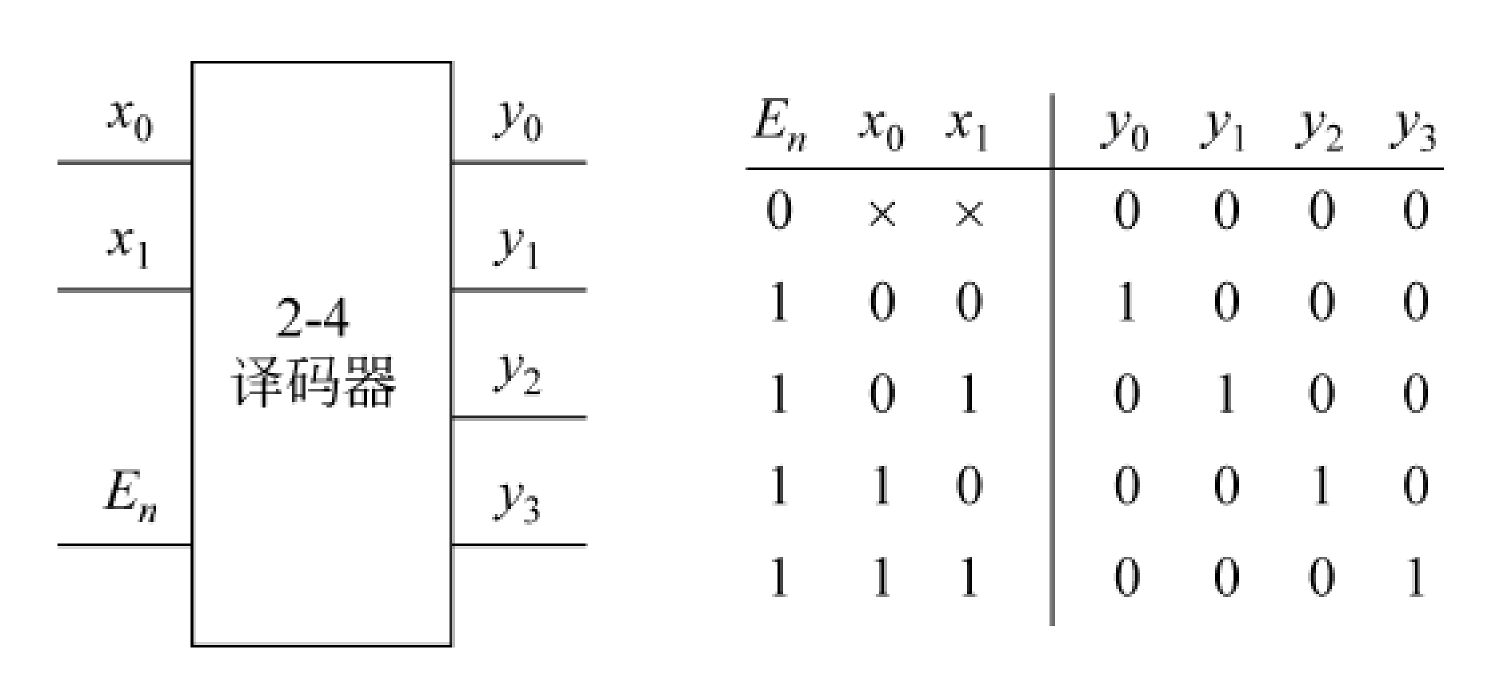
图 3-33　n 位输入，2^n 位输出的译码器

本次实验首先具体介绍 2-4 译码器和 3-8 译码器的工作原理，学习译码器的设计，同时学习一种常见的显示器件——七段 LED 数码管的使用。

3.2.1　2-4 译码器

2-4 译码器(图 3-34 所示)，它的输入信号是 x_0 和 x_1，对每一个二进制输入信号 x_0 和 x_1 进行译码，译码输出是在 y_0、y_1、y_2 和 y_3 4 位中选择 1 位输出使其有效。译码器的输出可以设计成高电平有效或者低电平有效，在本例中采用的是高电平有效。

2-4 译码器的逻辑电路图如图 3-35 所示。



E_n	x_0	x_1	y_0	y_1	y_2	y_3
0	×	×	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

图 3-34　2-4 译码器

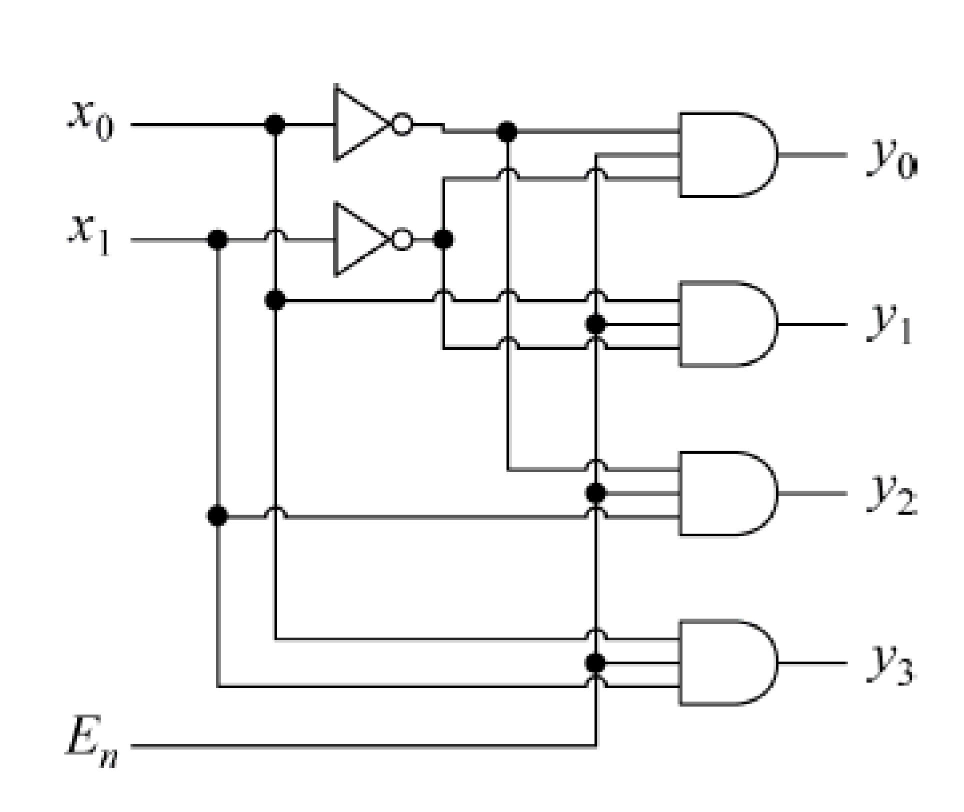


图 3-35　2-4 译码器逻辑电路

可以在 Quartus Ⅱ 编译器中直接画出 2-4 译码器的逻辑电路图，编译后生成二进制文件，下载至开发板中，实现 2-4 译码器。也可以采用硬件描述语言来实现 2-4 译码器电路。下面给出 2-4 译码器的程序清单。

程序清单 3-3

```
module decode24(x,en,y);
input   [1:0] x;
input   en;
output reg [3:0]y;
always @ (x or en)
     if (en) begin
        case (x)
                 2'd0 : y= 4'b0001;
                 2'd1 : y= 4'b0010;
                 2'd2 : y= 4'b0100;
                 2'd3 : y= 4'b1000;
        endcase
                end
   else   y= 4'b0000;
endmodule
```

程序清单 3-3 中使用了 if-else 语句,if-else 语句是 Verilog 语言中常用的语句。if-else 语句中,if 后面是一个要测试的条件表达式,如果满足条件,则执行后面的过程语句。如果要执行的语句不止一条,要用 begin-end 语句将要执行的所有语句“括”起来。一般情况下 if 和 else 配合使用,如果测试条件的值为假,则执行 else 后的过程语句。else 语句可以省略,但是,这时综合器就会综合出一个锁存器,用来保存不满足测试条件时在过程语句中被赋值的变量的过去值,在组合逻辑电路中,这是我们不愿意看到的。避免综合出锁存器的方法,就是保证每一个 if 语句都有一个 else 语句。

程序中还用到了数值字串(literal)“*n*'Bdd…d”,其中 *n* 是字串的位数,用十进制表示,这里字串的位数是“dd…d”这个值存放在机器中(二进制)所用的位数,而不是其用“B”进制表示的位数。“B”是指定基数的单个字母,可以是 b(二进制),o(八进制),d(十进制,可省略)和 h(十六进制)。“dd…d”是此数值字串,用“B”进制表示的值。

输入设计好的程序代码后,对工程进行分析与综合,在分析与综合后,可以查看分析综合的结果是否与所设想中的设计一致,利用 Quartus Ⅱ 的 RTL Viewer 和 Technology Map Viewer 两个工具可以实现这一目的。

用 Quartus Ⅱ 的 Netlist Viewers 工具查看编译后生成的电路。打开方法:选择菜单项“Tools→Netlist Viewers→RTL Viewer”或者“Tools→Netlist Viewers→ Technology Map Viewer”,如图 3-36 所示。

RTL Viewer 是指寄存器传输级原理图,用户可以根据此图查看根据设计代码产生的寄存器传输级图是否满足设计需求,同时也可以查看工程的层次结构列表、整个设计的实例、基本单元、引脚和网络。而 Technology Map Viewer 提供的是设计的底级或基元级专用技术原理表征,它展示的是 RTL 视图向具体器件进行结构映射的结果,根据这个图

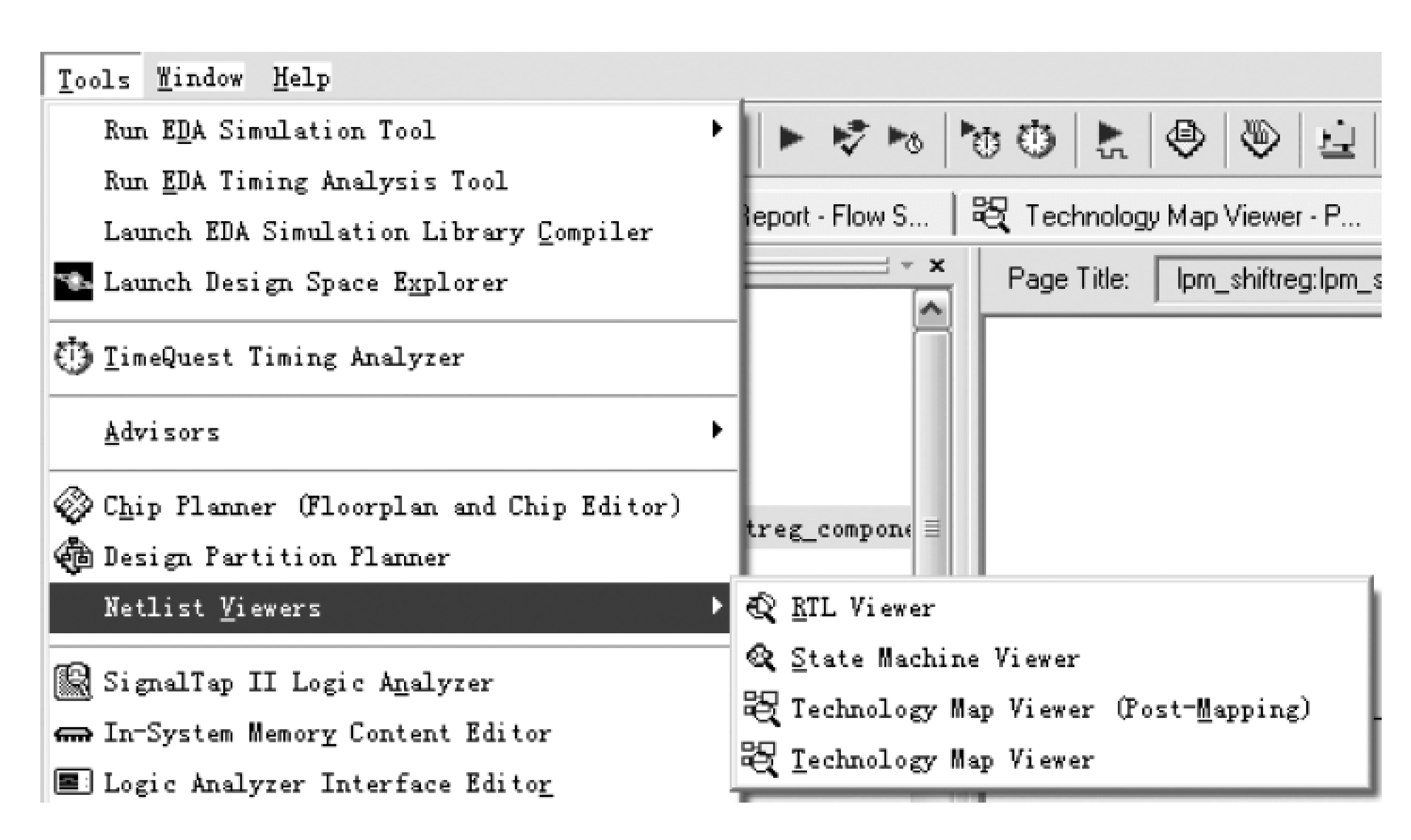


图 3-36　查看 Netlist Viewer

我们可以查看设计映射到具体 FPGA 的 LUT 的工作方式。本例中的 Technology Map Viewer 图，如图 3-37 所示。

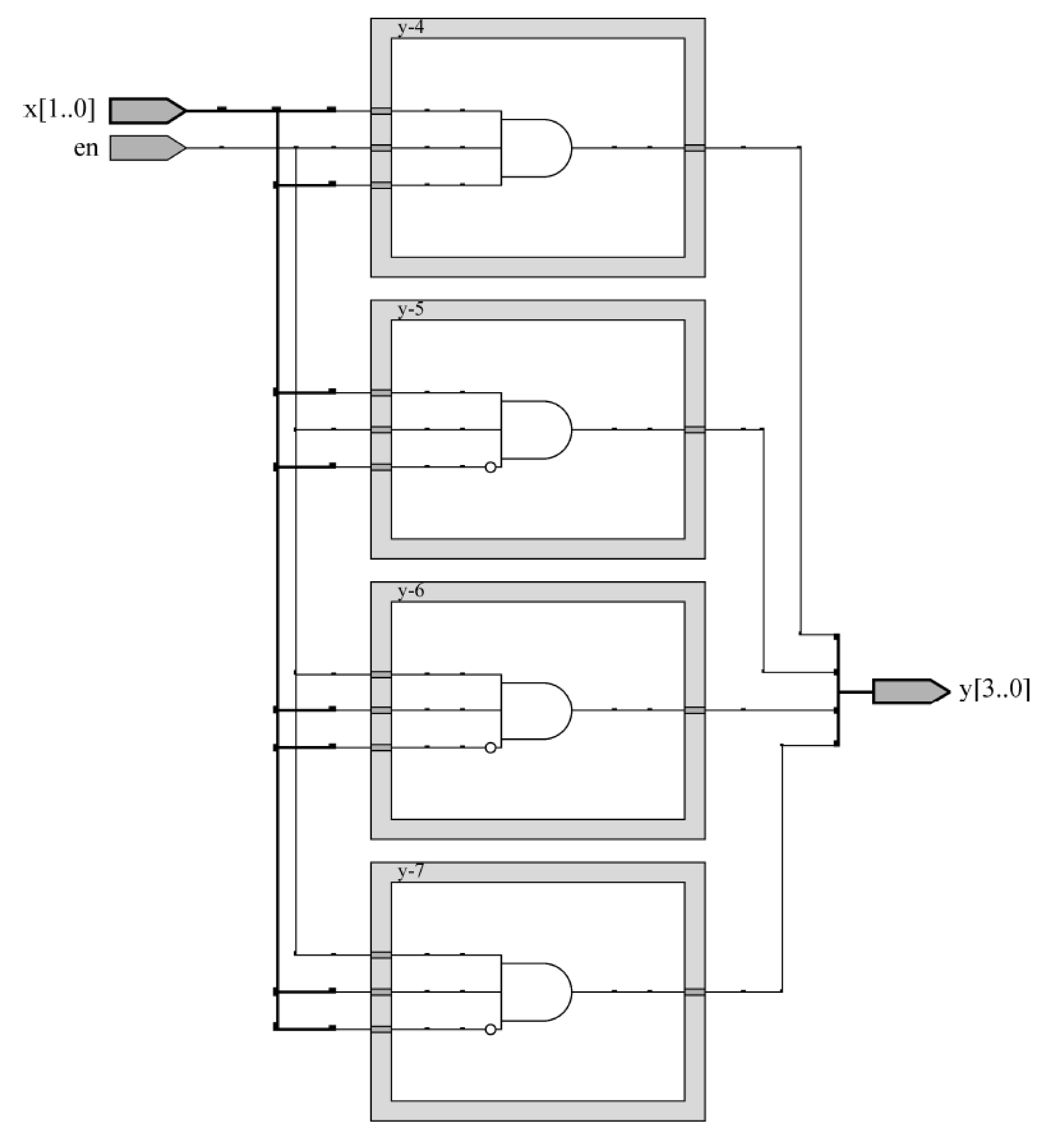


图 3-37　2-4 译码器的 Technology Map Viewer

将 2-4 译码器的 Technology Map Viewer 和先前设计的电路原理图对照发现，满足设计原理图。

对所设计电路进行功能仿真，如图 3-38 所示，分析时序图发现，结果和真值表一致。

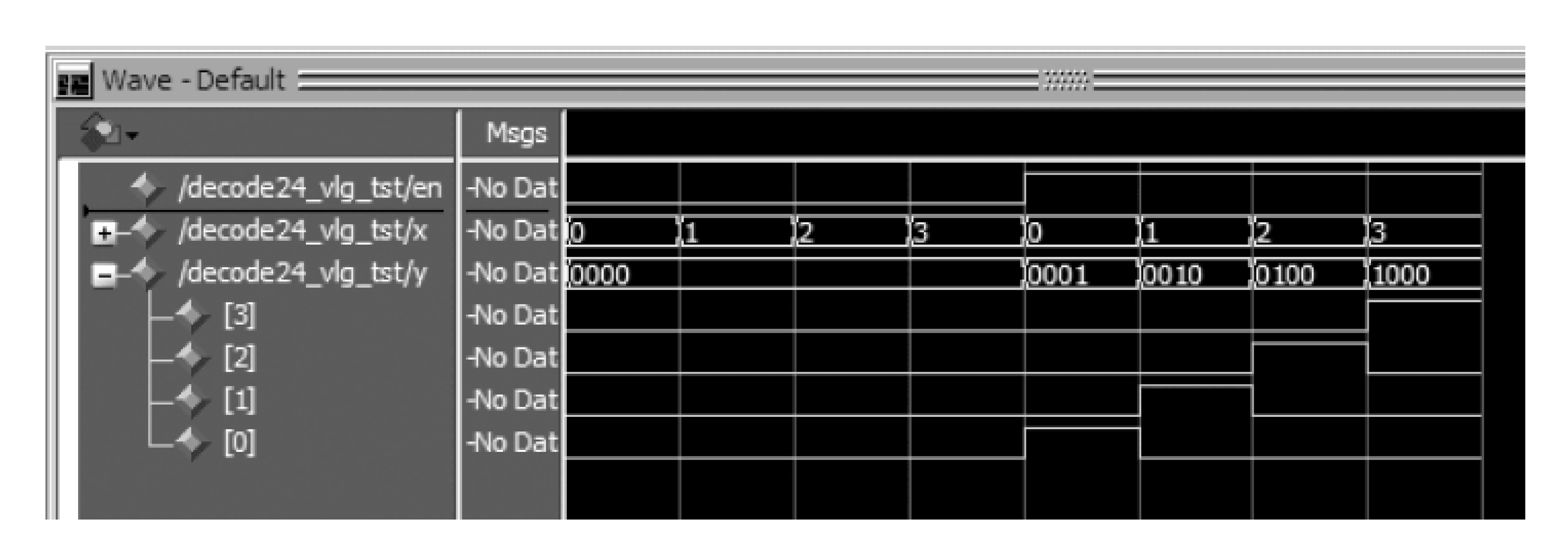


图 3-38　2-4 译码器功能仿真

3.2.2　3-8 译码器

3-8 译码器和 2-4 译码器在原理上是相同的，本实验的目的是学习使用已有的文件来实现新的工程。

3-8 译码器的真值表如图 3-39 所示。

E_n	x_2	x_1	x_0	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
0	×	×	×	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

图 3-39　3-8 译码器真值表

建立新的工程 decode38。在 Add Files 界面，单击“File name”后面的“…”按钮，找到先前已经设计好的文件 decode24. v 的路径，如图 3-40 所示。

选择文件，将其加入本工程，如图 3-41 所示。

加入文件，如图 3-42 所示。

完成新建工程，在工程的项目导航中，“file”栏中会出现刚才加入的文件名，双击文件名就可显示文件内容，如图 3-43 所示。

在 QuartusⅡ 中单击 File 菜单，在“Create/Update”中单击“Create Symbol file for Current project”选项，如图 3-44 所示。这时 QuartusⅡ 会为这个文件生成一个只带此元件外围接口的符号图形。

在 QuartusⅡ 中新建图形设计文件。单击 File 菜单下的“New”选项，选择“Block Diagram/Schematic File”文件。

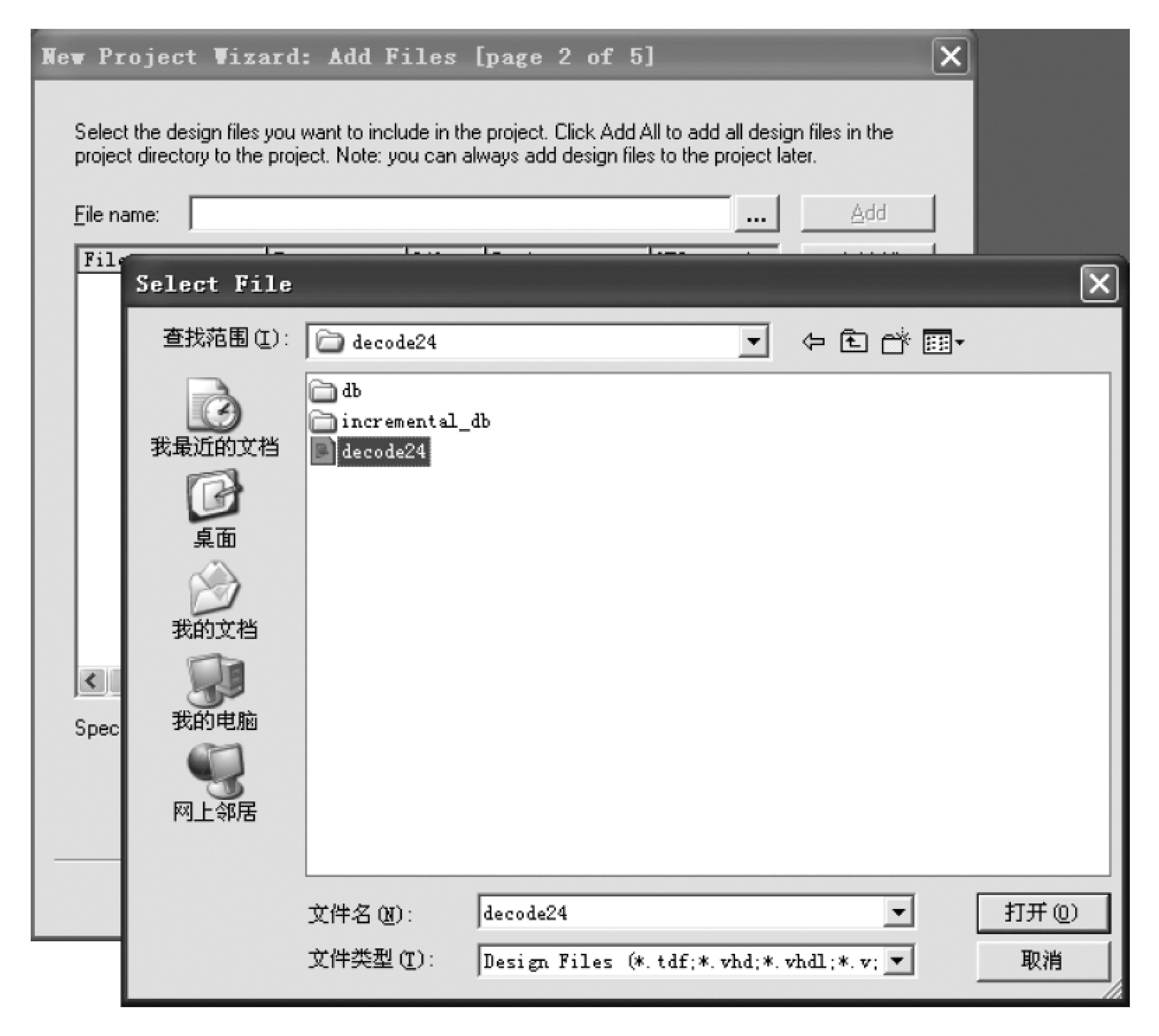


图 3-40　要加入的文件路径

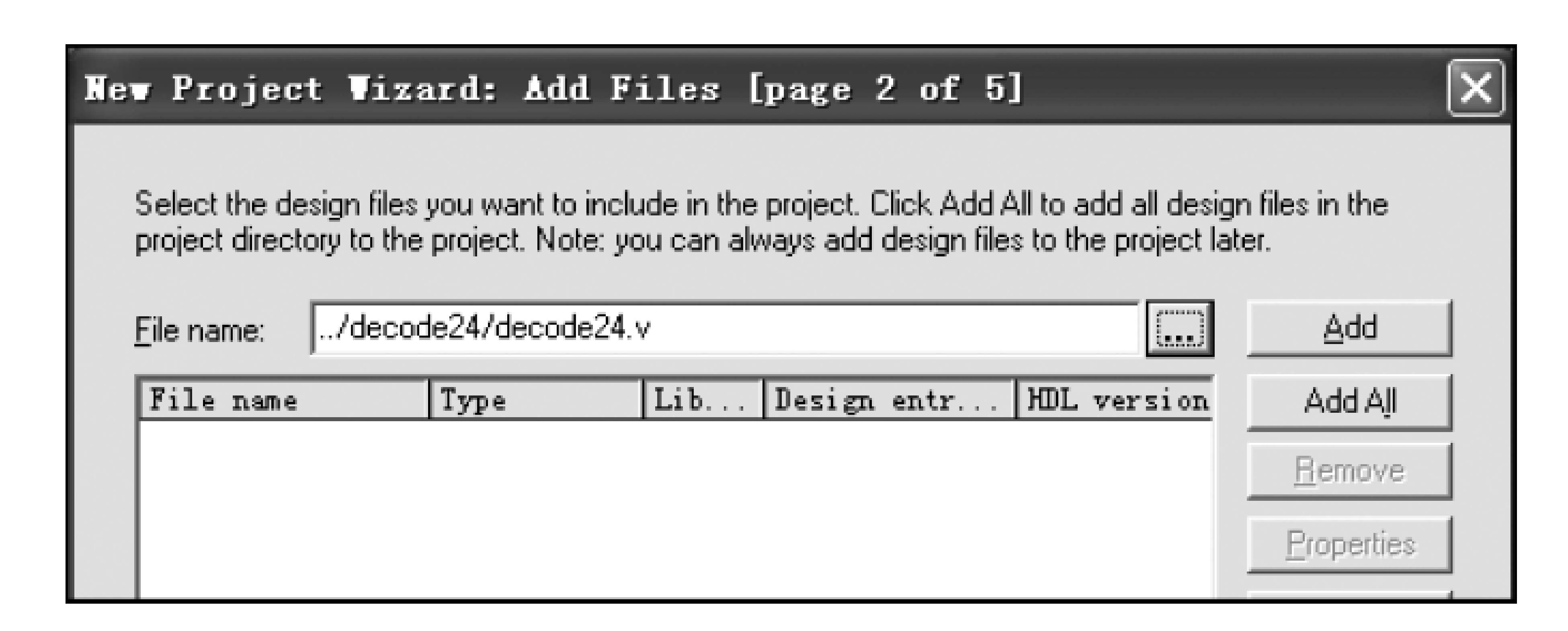


图 3-41　选择要加入的文件

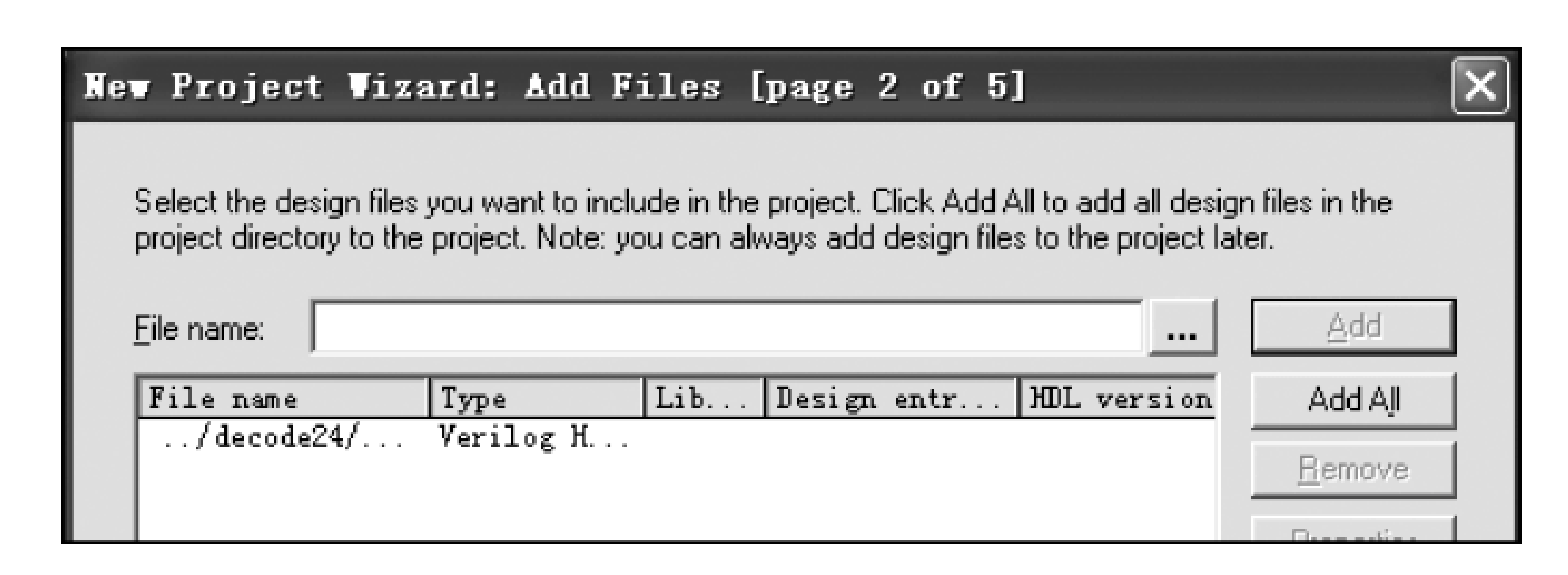


图 3-42　加入文件

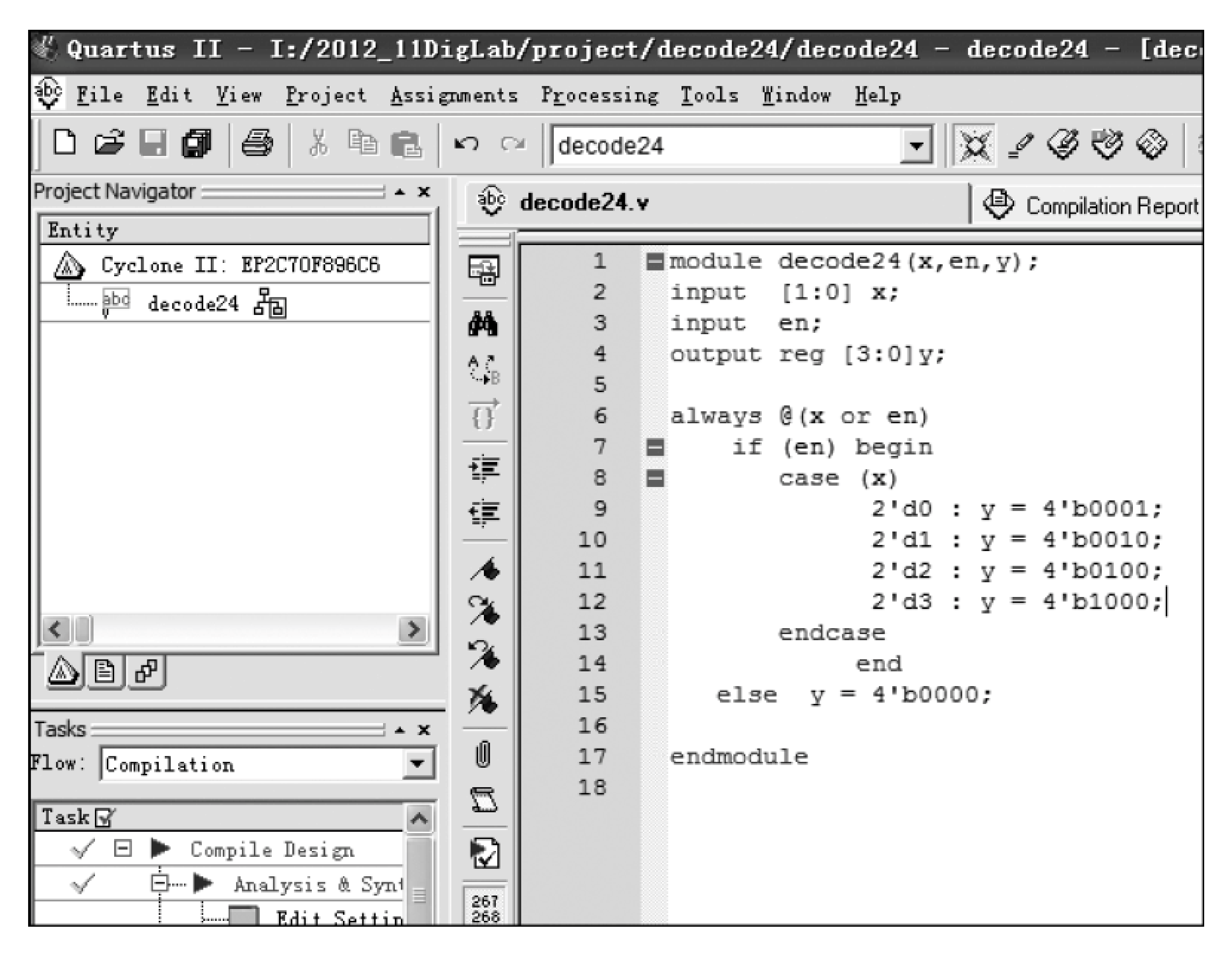


图 3-43 加入的文件

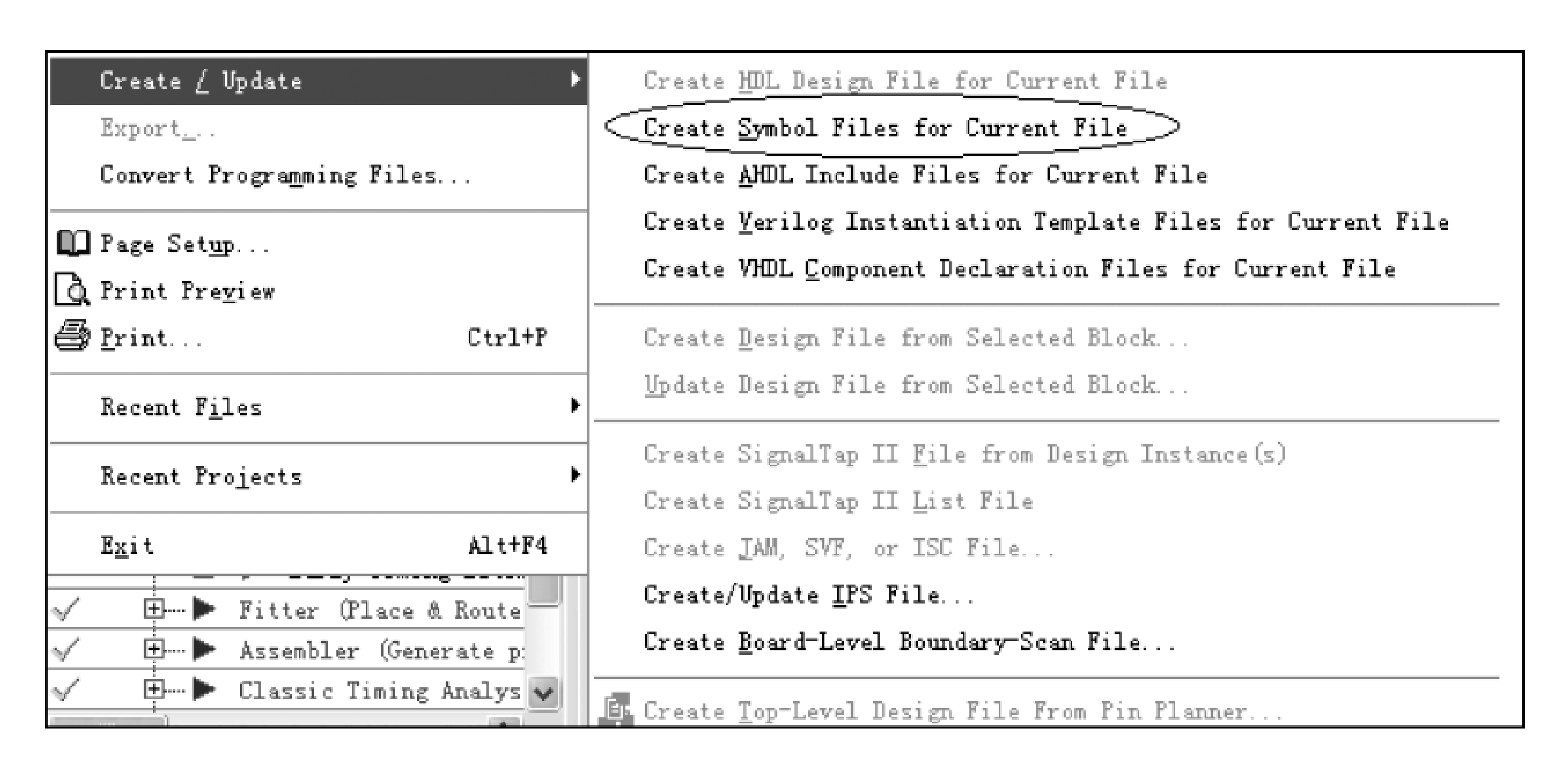


图 3-44 产生图形符号

在新建的图形设计文件中会看到有很多小点，在随意的一个地方双击鼠标左键，会弹出如图 3-45 所示的界面。

展开 Project 会出现上一步生成的 2-4 译码器 decode24 的符号图形，单击 decode24，则在右侧栏同时会显示它的顶层图形。这个图形就是 QuartusⅡ为 HDL 文件生成的图形设计，如图 3-46 所示。

单击“OK”按钮，把图形符号放进 Quartus Ⅱ 的文件编辑区，从这个图形符号可以看出，上一个工程生成的器件端口部分全部显示出来，左边的是输入端口，右侧的是输出端口。

在图形编辑窗口加入两个 2-4 译码器，再加入简单的门电路，将其构成一个 3-8 译码

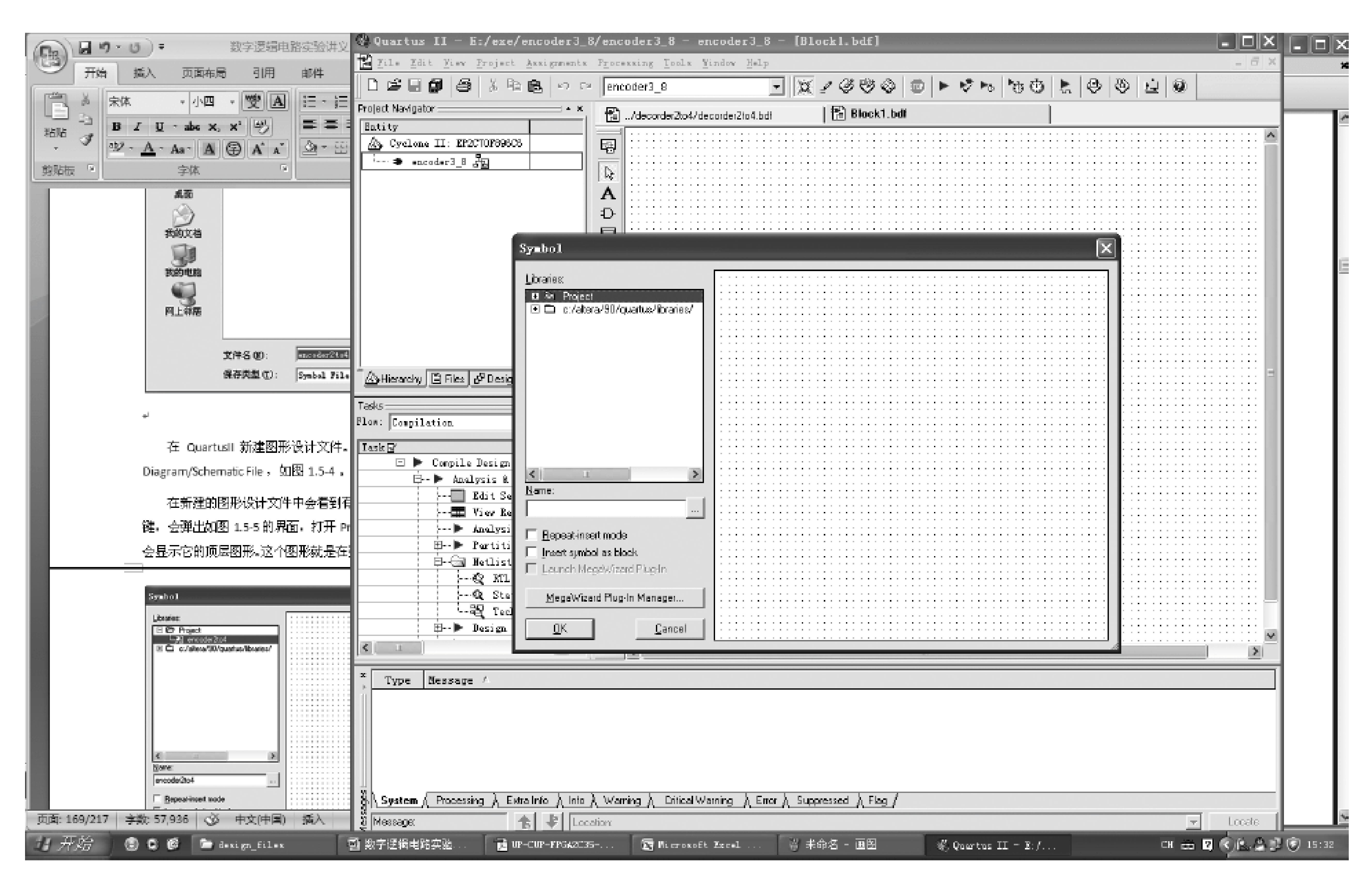


图 3-45　添加图形符号

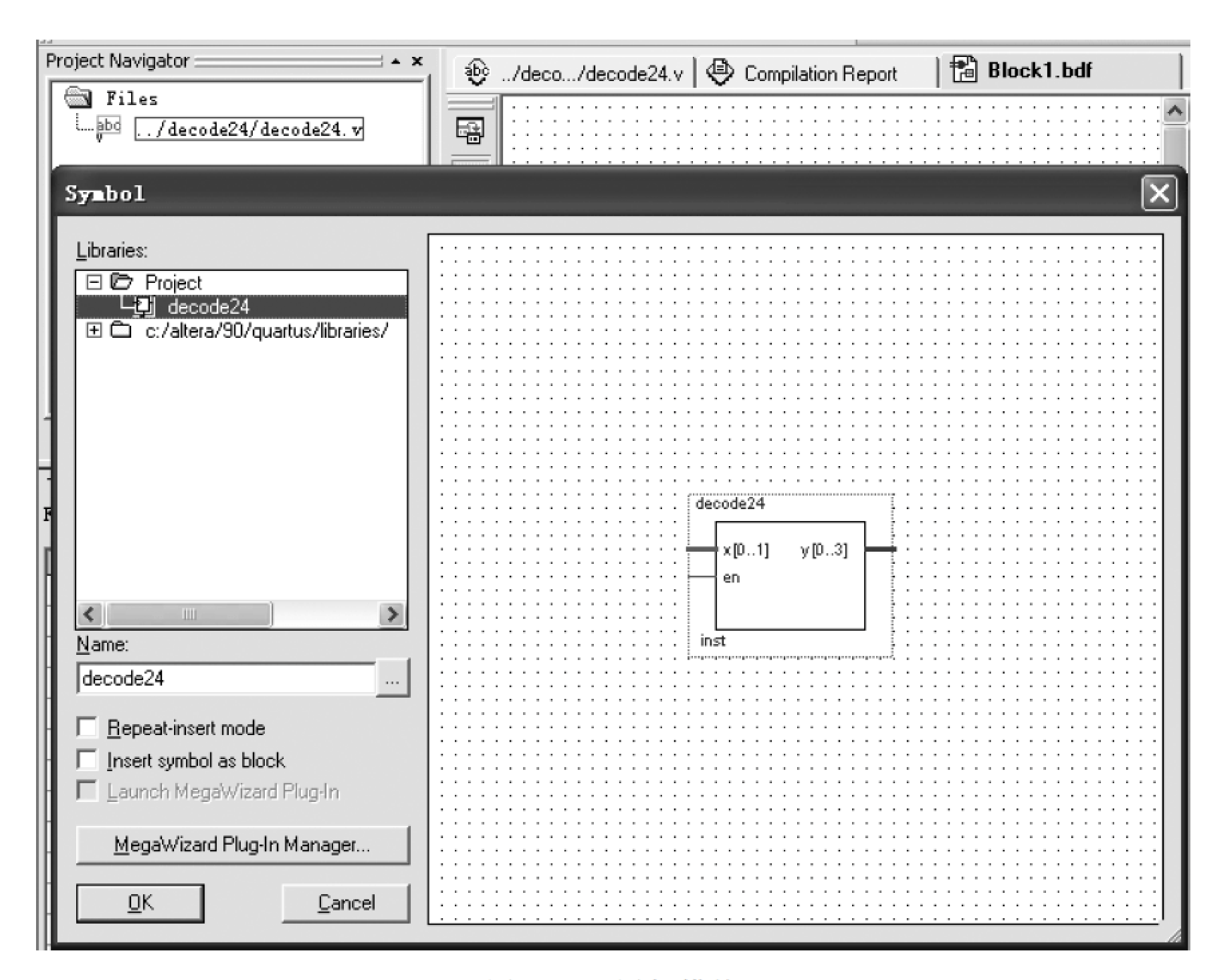


图 3-46　添加模块

器，如图 3-47 所示。

完成 3-8 译码器的设计后，对工程进行分析与综合，分析与综合后重新生成“.v”文件，进行功能仿真，仿真结果如图 3-48 所示。

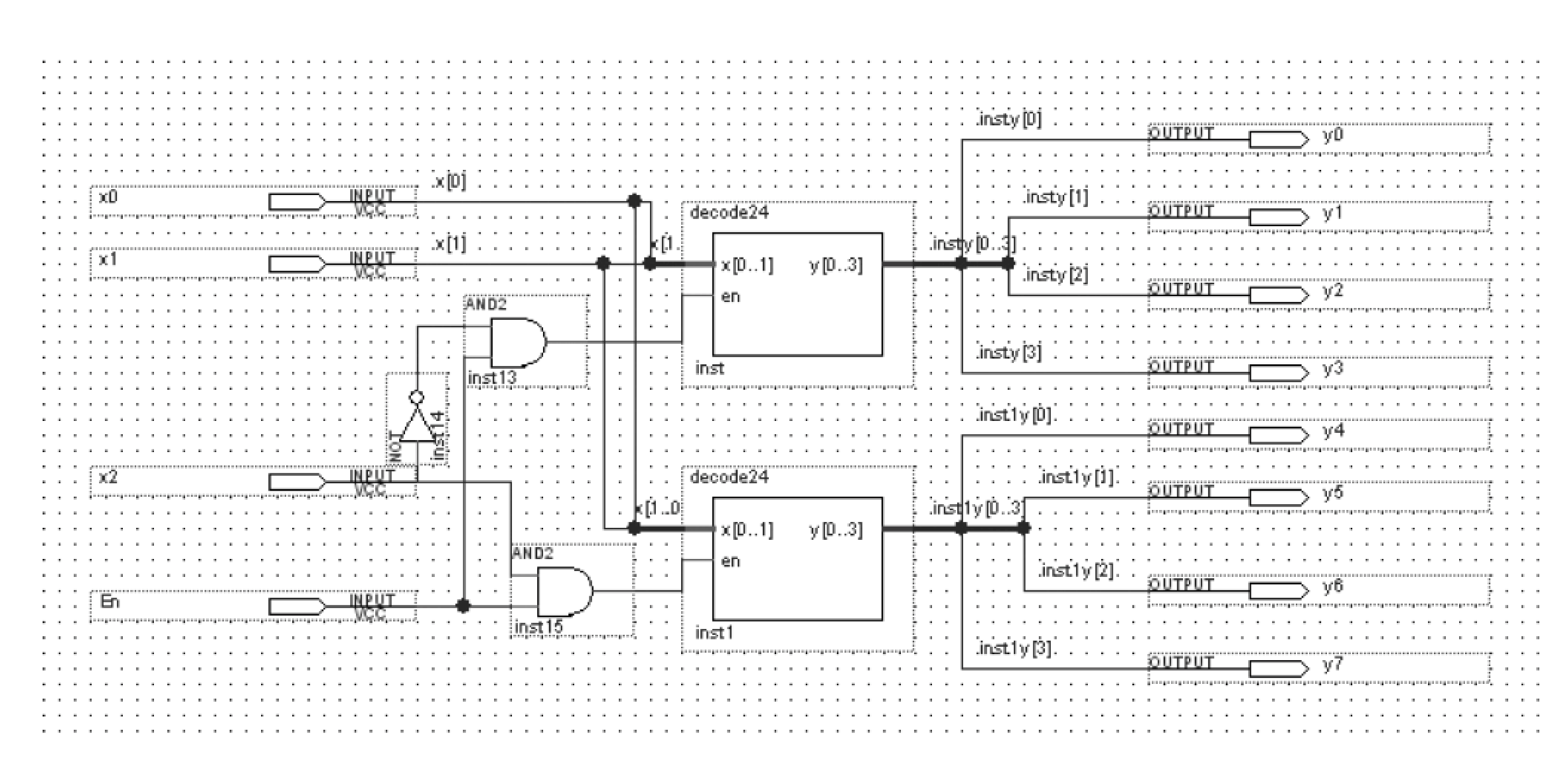


图 3-47　3-8 译码器

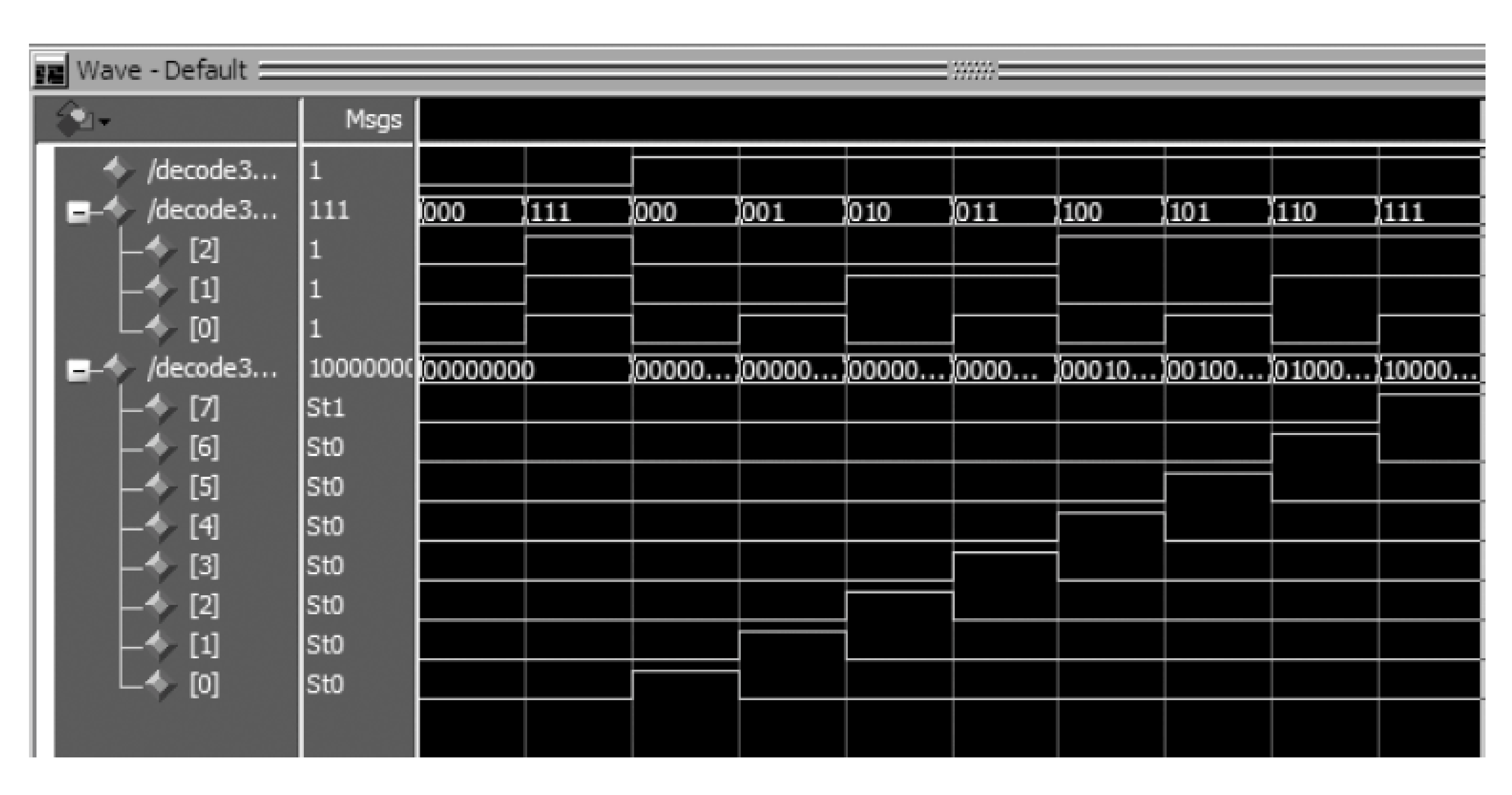


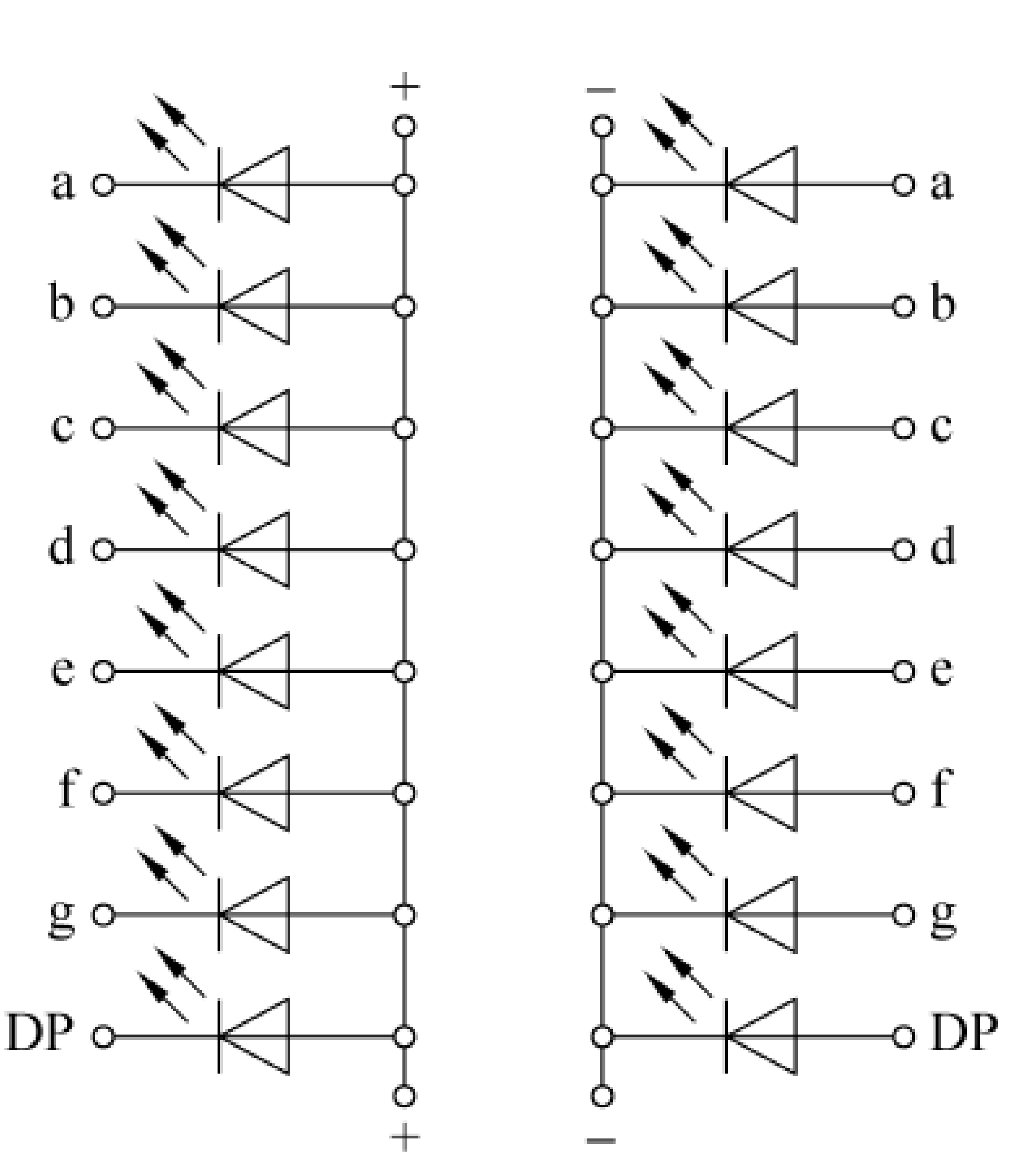
图 3-48　3-8 译码器功能仿真图

添加引脚，完成工程，全编译，下载到 FPGA 上验证设计的正确性。

3.2.3　实验内容

七段 LED 数码管是一种常用的显示元件，常应用于手表、计算器等仪器中，用于显示十进制数值。七段显示器采用七段译码，一般情况下，其输入为 4 位的 BCD 码，输出为7 位的编码，用于驱动七段显示器的不同位，以显示出不同的数字。

图 3-49 是数码管的原理图，图中在七段数码管的基础上又加了一个小数点 DP，一共是 8 个 LED 数码管。数码管分为共阴极和共阳极两种类型，共阴极 LED 数码管就是将 8 个 LED 的阴极连在一起，让其接低电平，这样给任何一个



(a) 共阳极LED数码管　(b) 共阴极LED数码管

图 3-49　LED 数码管原理图

LED 的另一端高电平，它就能点亮。而共阳极 LED 数码管就是将 8 个 LED 的阳极连在一起，让其接高电平，这样，给任何一个 LED 的另一端低电平，它就能点亮。

DE2-70 开发平台上有 8 个带小数点的七段数码管，图 3-50 是 DE2-70 开发平台数据手册中提供的开发板上的数码管的电路连接图，由电路可以看出，这些数码管是共阳极连接方式，数码管的各个引脚和 FPGA 的引脚相连，如果给数码管的某个引脚输入低电平则该数码管会被点亮，若输入高电平则该数码管变暗。

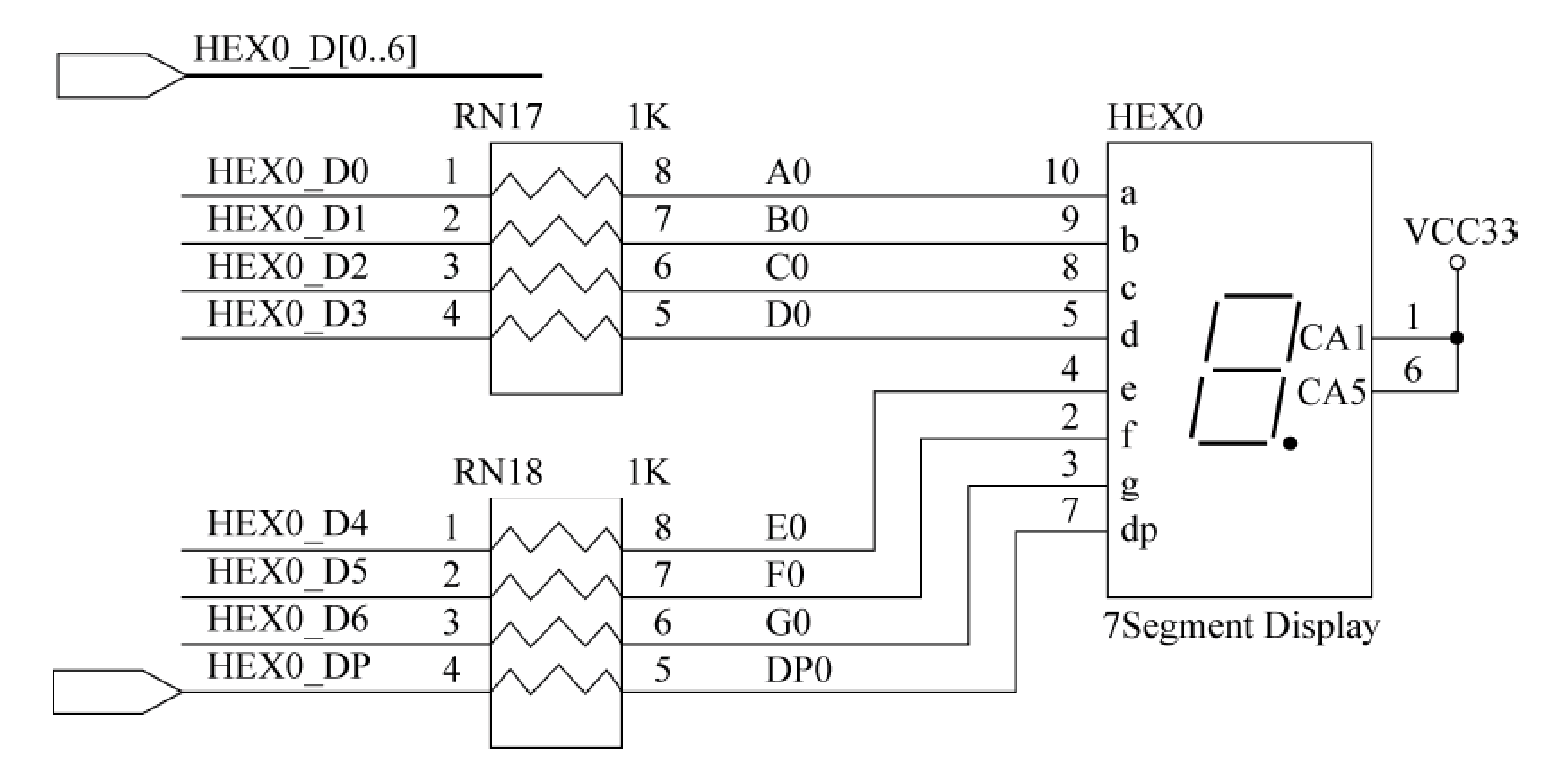


图 3-50　数码管电路图

数码管的各段按英文字母 a～g 编码，如图 3-51 所示。

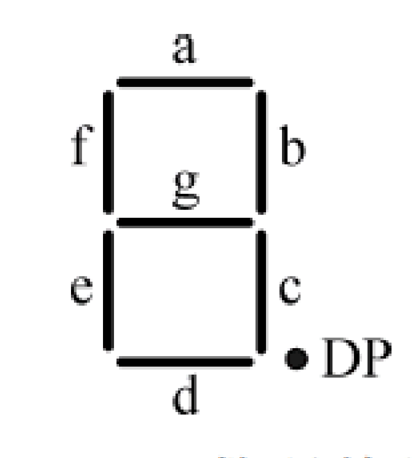


图 3-51　数码管编号

DE2-70 平台上的 8 个七段数码管分别标识为 oHEX0～oHEX7，数码管的每一段都分别被标以 a～g 的字母并由逻辑 0 点亮，在平台配套的引脚配置文件中，数码管的各段的名称分别为（以 oHEX0 为例）：oHEX0_D[0]～oHEX0_D[6]。另外，每个数码管都带一个"点"，这个"点"名称为（以 oHEX0 为例）：oHEX0_DP，也是逻辑 0 点亮，在使用中请注意。

本实验的要求是：

1. 数码输出显示对应数值

在 oHEX3-oHEX0 上显示 iSW[15]～iSW[0]所对应的数值：iSW[15]～iSW[12]，iSW[11]～iSW[8]，iSW[7]～iSW[4]和 iSW[3]～iSW[0]分别对应 oHEX3、oHEX2、oHEX1 和 oHEX0。在数码管上只显示 0～9，当拨动开关表示的数字在 1010～1111 之间时，没有显示输出。例如，当 iSW[3]～iSW[0]输入"0000"时，数码管 oHEX0 上显示"0"，此时数码管 oHEX0 上编号为 a、b、c、d、e 和 f 的发光二极管亮，其他发光二极管暗，即给编号为 a、b、c、d、e 和 f 的发光二极管的引脚输入低电平"0"，其他发光二极管的引脚则输入高电平"1"。

分析实验内容，画出真值表，形式如表 3-1 所示。

分别推导出 oHEX0_D[6]～oHEX0_D[0]与 iSW[3]～iSW[0]、oHEX1_D[6]～oHEX1_D[0]与 iSW[7]～iSW[4]、oHEX2_D[6]～oHEX2_D[0]与 iSW[11]～iSW[8]和 oHEX3_D[6]～oHEX3_D[0]与 iSW[15]～iSW[12]之间的逻辑表达式，设计电路，完

表 3-1 数码管显示真值表

iSW3	iSW2	iSW1	iSW0	oHEX0[6]	oHEX0[5]	oHEX0[4]	oHEX0[3]	oHEX0[2]	oHEX0[1]	oHEX0[0]
0	0	0	0	1	0	0	0	0	0	0

成后下载至 FPGA，观察显示的结果是否满足设计要求。

本实验要求先设计一个子模块，完成数码管的显示代码，在顶层实体模块中调用此子模块逐个点亮数码管。以下是本实验的模块结构：

```
//在数码管上显示输入的二进制值
//输入端为 iSW,输出端为 oHEX
module part1(iSW, oHEX3, oHEX2, oHEX1, oHEX0);     //顶层实体模块
    …
                      //驱动七段数码管显示十进制数
    bcd7seg digit0(iSW[3:0], oHEX0);
    …
endmodule
                //数码管显示模块
module bcd7seg(B, H);
    …
    case(B)
    …
    4'd0 H=7'b1000000
…
endmodule
```

2. 将 4 位二进制值转换为 2 位十进制显示

将 4 位二进制数，如 iSW[3]～iSW[0]转换成 2 位十进制的值显示在 oHEX0～oHEX1 上，输入的二进制值与输出的十进制值之间的关系如表 3-2 所示。

表 3-2 二进制与十进制的转换关系

iSW3	iSW2	iSW1	iSW0	oHEX1	oHEX0
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	2
0	0	1	1	0	3

续表

iSW3	iSW2	iSW1	iSW0	oHEX1	oHEX0
0	1	0	0	0	4
0	1	0	1	0	5
0	1	1	0	0	6
0	1	1	1	0	7
1	0	0	0	0	8
1	0	0	1	0	9
1	0	1	0	1	0
1	0	1	1	1	1
1	1	0	0	1	2
1	1	0	1	1	3
1	1	1	0	1	4
1	1	1	1	1	5

请设计电路，完成上述功能。

3.3　编码器的设计

编码器是一种与译码器功能相反的逻辑电路，编码器的输出编码比其输入编码位数少。

常用的二进制编码器把来自于 2^n 条输入信号线的信息编码转换成 n 位二进制码，如图 3-52 所示。二进制编码器每次输入的 2^n 位信号中只能有一位为“1”，其余均为“0”，即独热码，编码器的输出端为一个二进制数，用来指示对应的哪一个位输入为“1”。

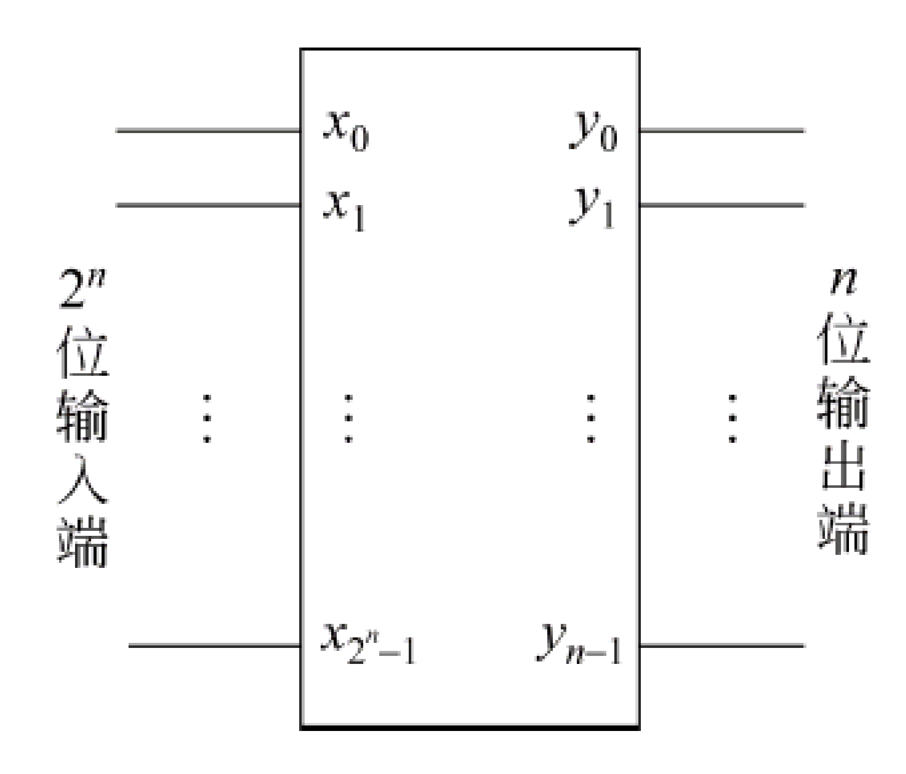


图 3-52　2^n 位输入、n 位输出的译码器

本实验的目的是熟悉常用的编码器 4-2 编码器的设计；学习 Quartus Ⅱ编译器中改变工程的顶层实体的编译方法；学习常用商用 MSI 8 输入优先编码器 74LS148 的设计，并学习利用 74LS148 扩展更多输入信号的优先编码器。

3.3.1　4-2 编码器

4-2 编码器即为 4 位输入、2 位输出的编码器（如表 3-3 所示），它的输入信号是 x_0、x_1、x_2 和 x_3，输出是 y_0 和 y_1。本例中采用独热码，每次输入中只有一位为“1”，对于有两位或者两位以上为“1”的情况未予考虑。请设计一个电路完成此 4-2 编码器。

表 3-3　4-2 编码器

x_3	x_2	x_1	x_0	y_1	y_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

建立工程，名为 bianma4_2，顶层实体名为 bianma4_2，如图 3-53 所示。

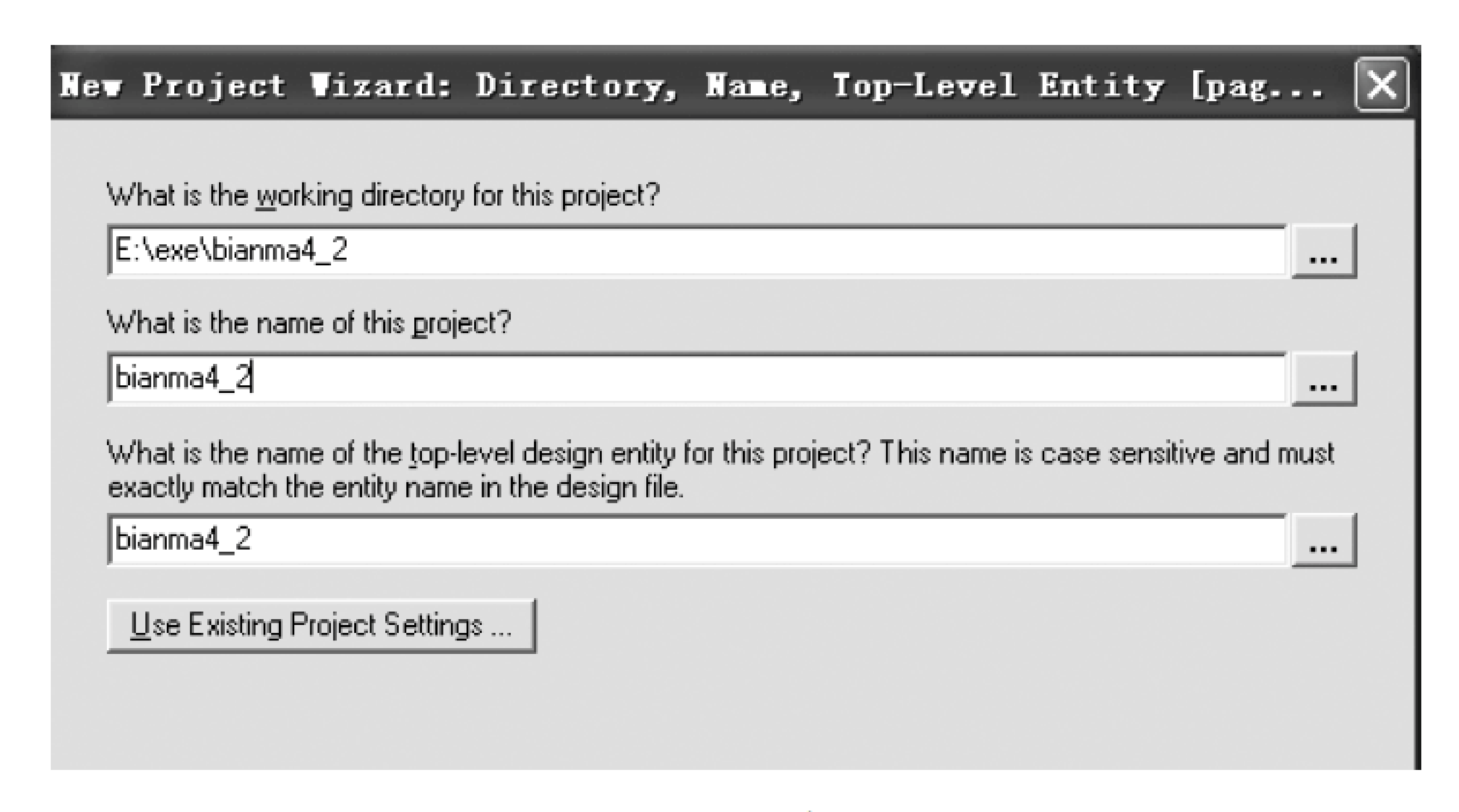


图 3-53　新建工程

本实验中，我们先设计一个 4-2 编码器模块，然后在工程的顶层实体文件中配置引脚，验证功能。

新建一个文件输入设计代码，保存为 encode4_2，如图 3-54 所示。

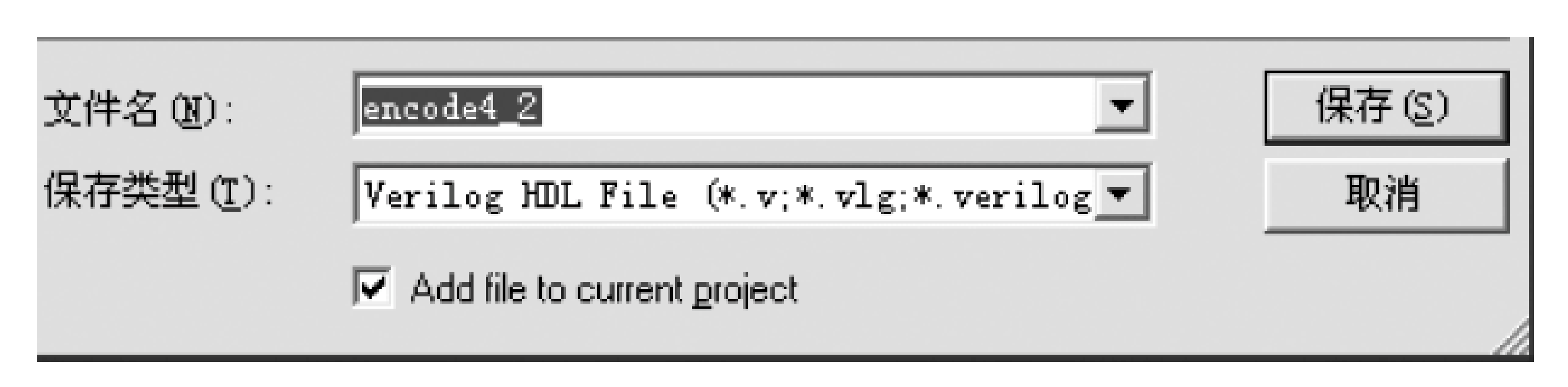


图 3-54　保存设计文件

此时，在项目导航的"File"栏，会出现文件 encode4_2，如图 3-55 所示。

在 encode4_2 文件上单击鼠标右键，选择菜单选项"Set as Top-Level Entity"，将此文件设置为顶层实体文件，如图 3-56 所示。

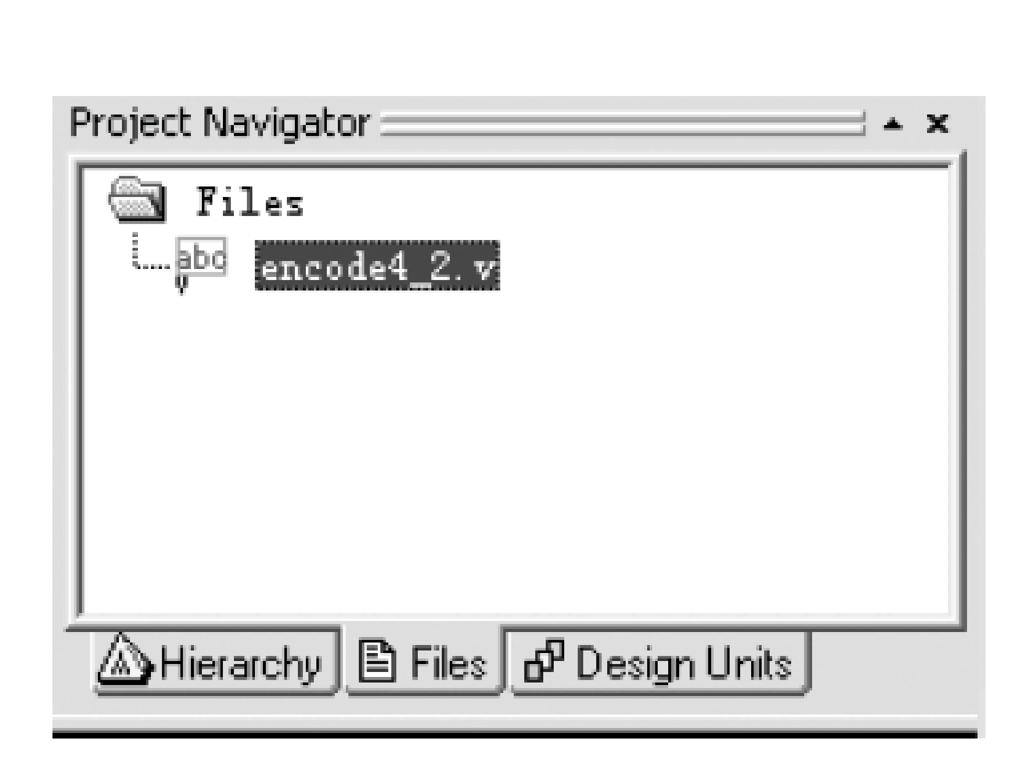


图 3-55　项目导航界面

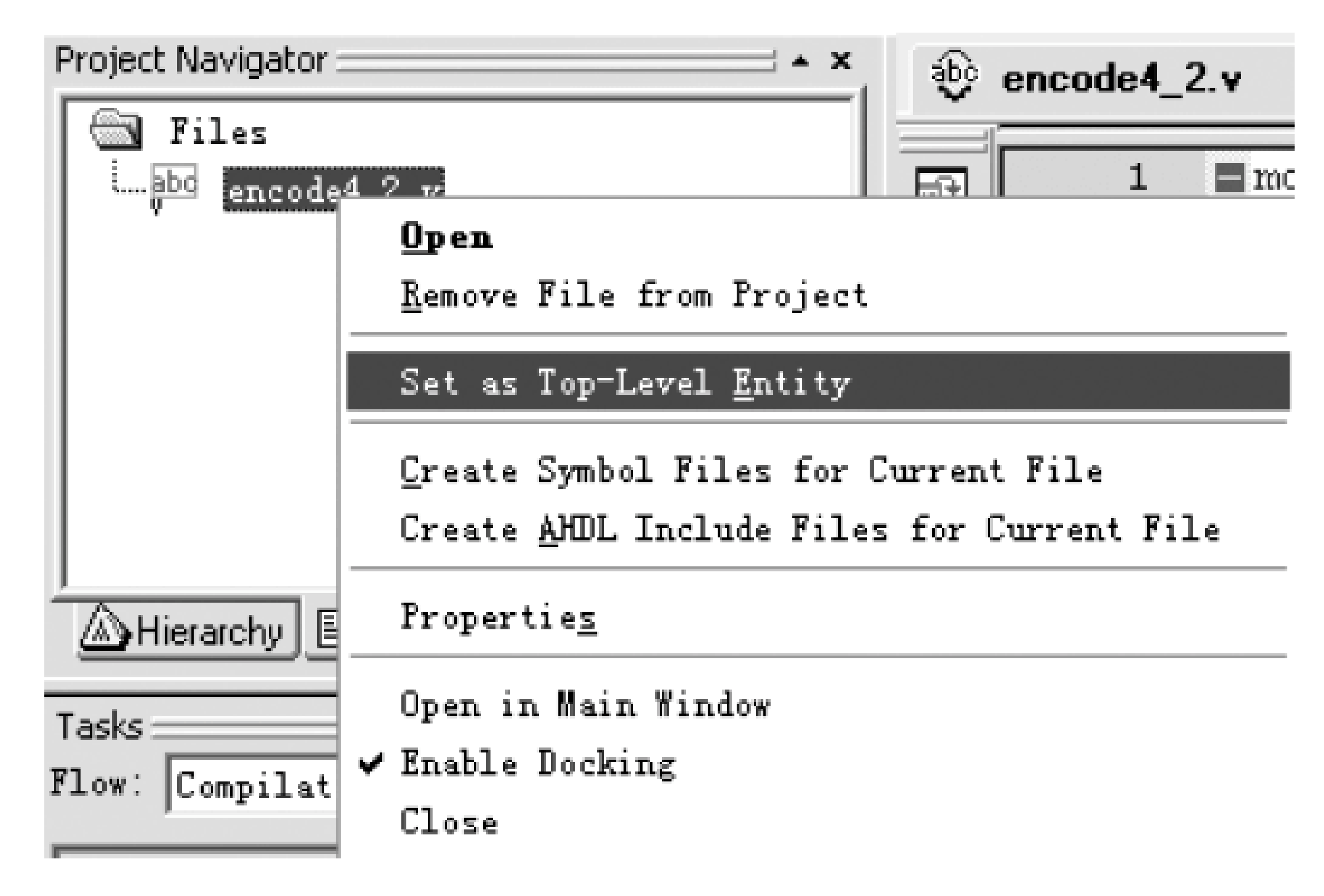


图 3-56　设置工程实体

对工程进行分析与综合，检查错误并修改错误。

仿真：新建仿真测试文件，默认生成名称和顶层实体文件名称相同的文件，如图 3-57 所示。

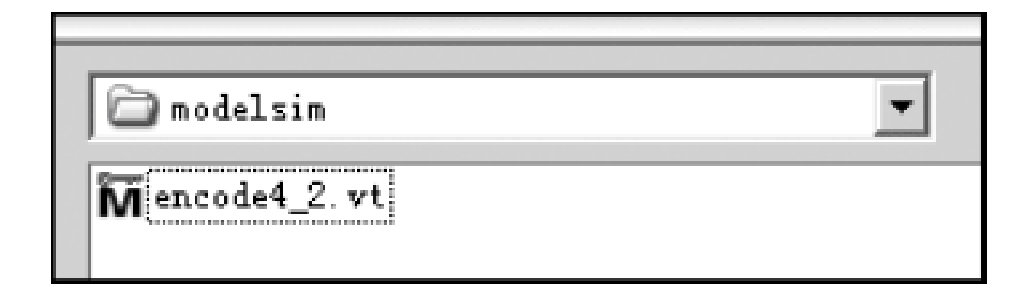


图 3-57　保存仿真文件

修改测试文件，设计仿真信号。

仿真，在对仿真进行设置时请注意，在设置仿真顶层实体名和添加仿真测试文件时，必须选择自己需要的文件进行设置和添加。单击"Assignments→Settings→Simulation → Compile test bench→Test Benches"，本例中，在"Test bench name"中应填入当前顶层实体的仿真测试代码"encode4_2_vlg_tst"，在"File name"栏中应选择"encode4_2.vt"，如图 3-58 所示。

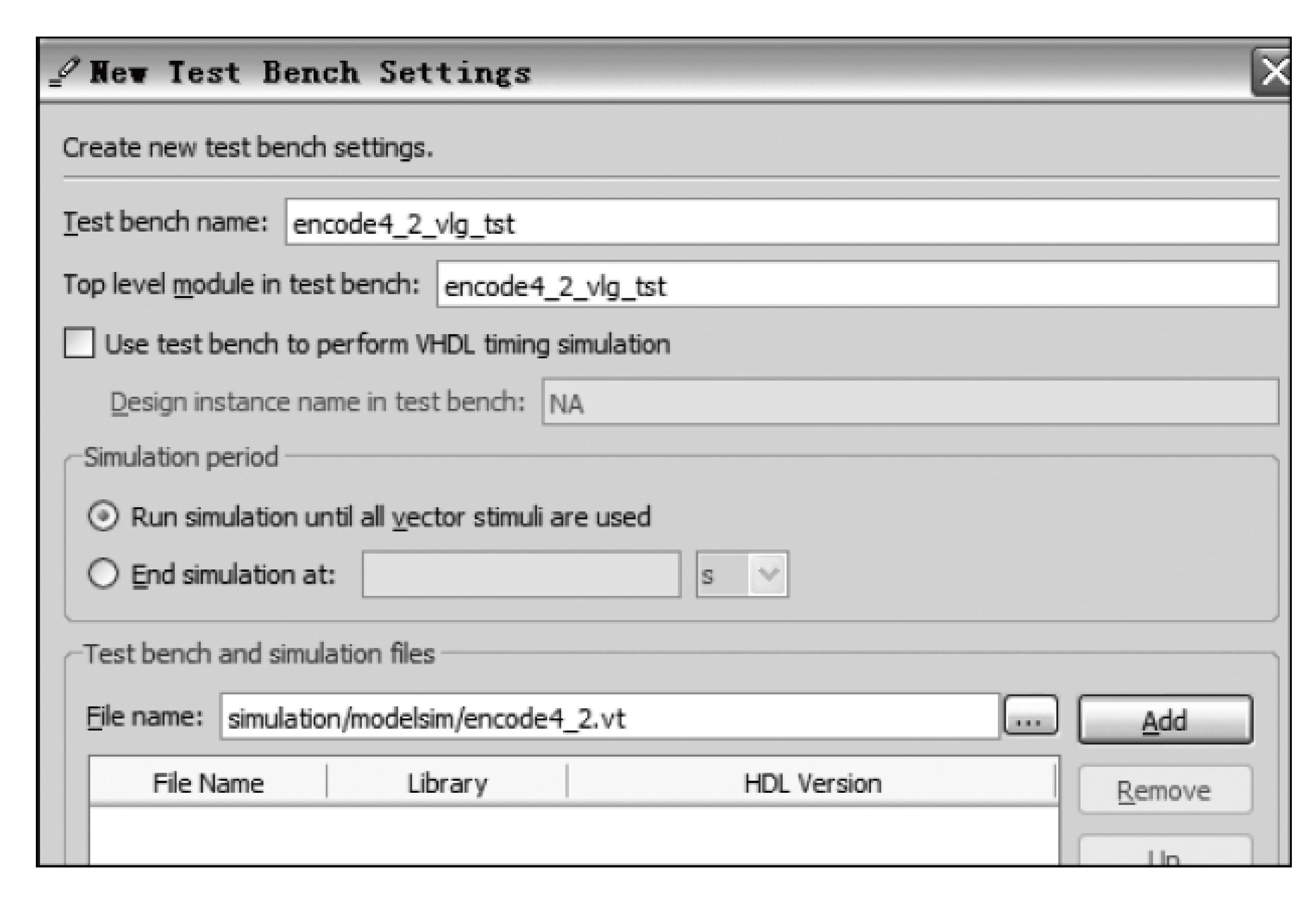


图 3-58　仿真设置

仿真并验证该功能模块正确后，为该文件产生模块框图，如图 3-59 所示。

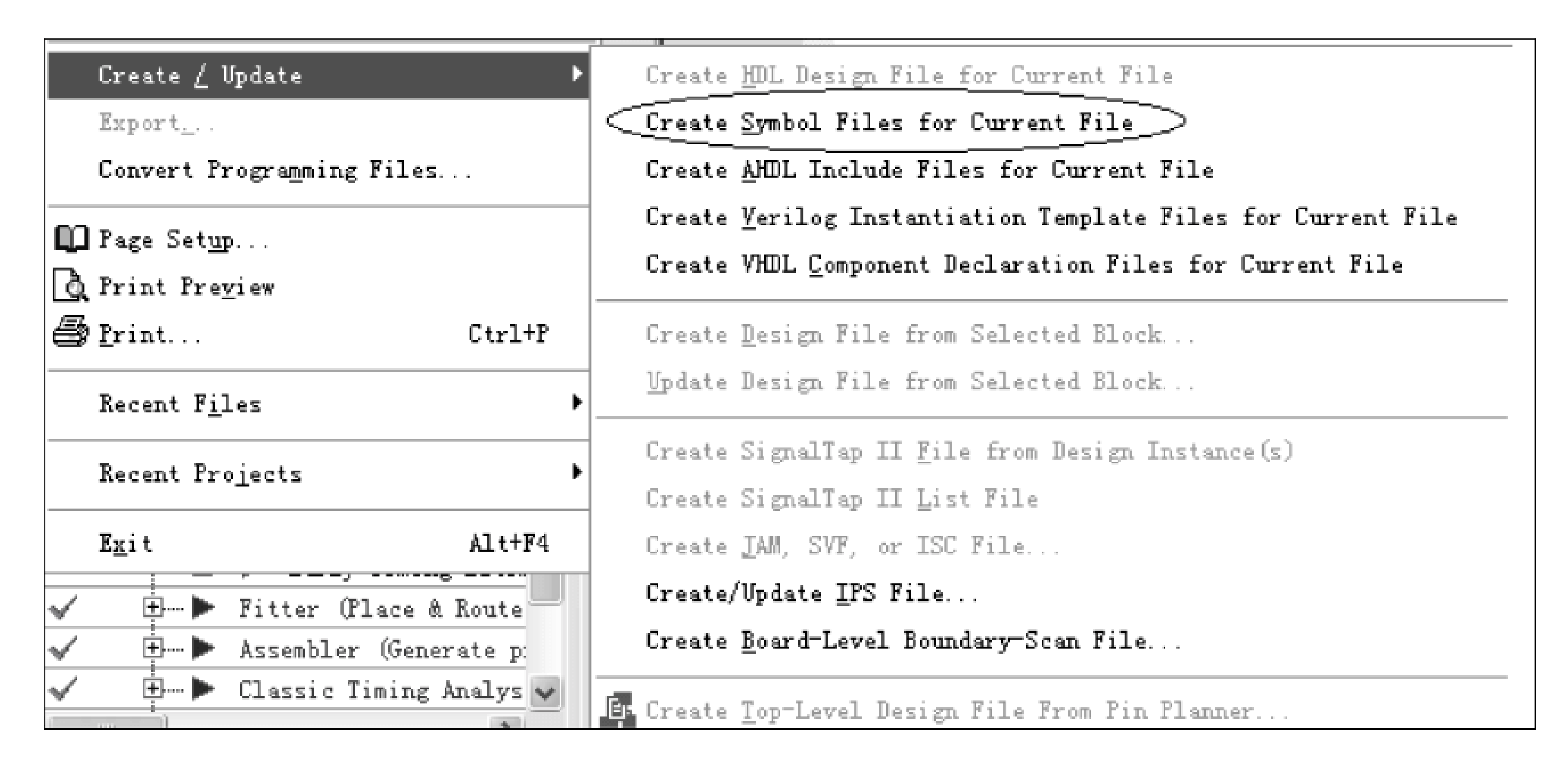


图 3-59　产生模块框图

新建"Block Diagram/Schematic File"文件，输入顶层实体文件。在图形编辑输入窗口的任意位置双击鼠标左键，将出现添加模块符号对话框，展开 Project 目录，可以看见刚才生成的 encode4_2 编码器，如图 3-60 所示。

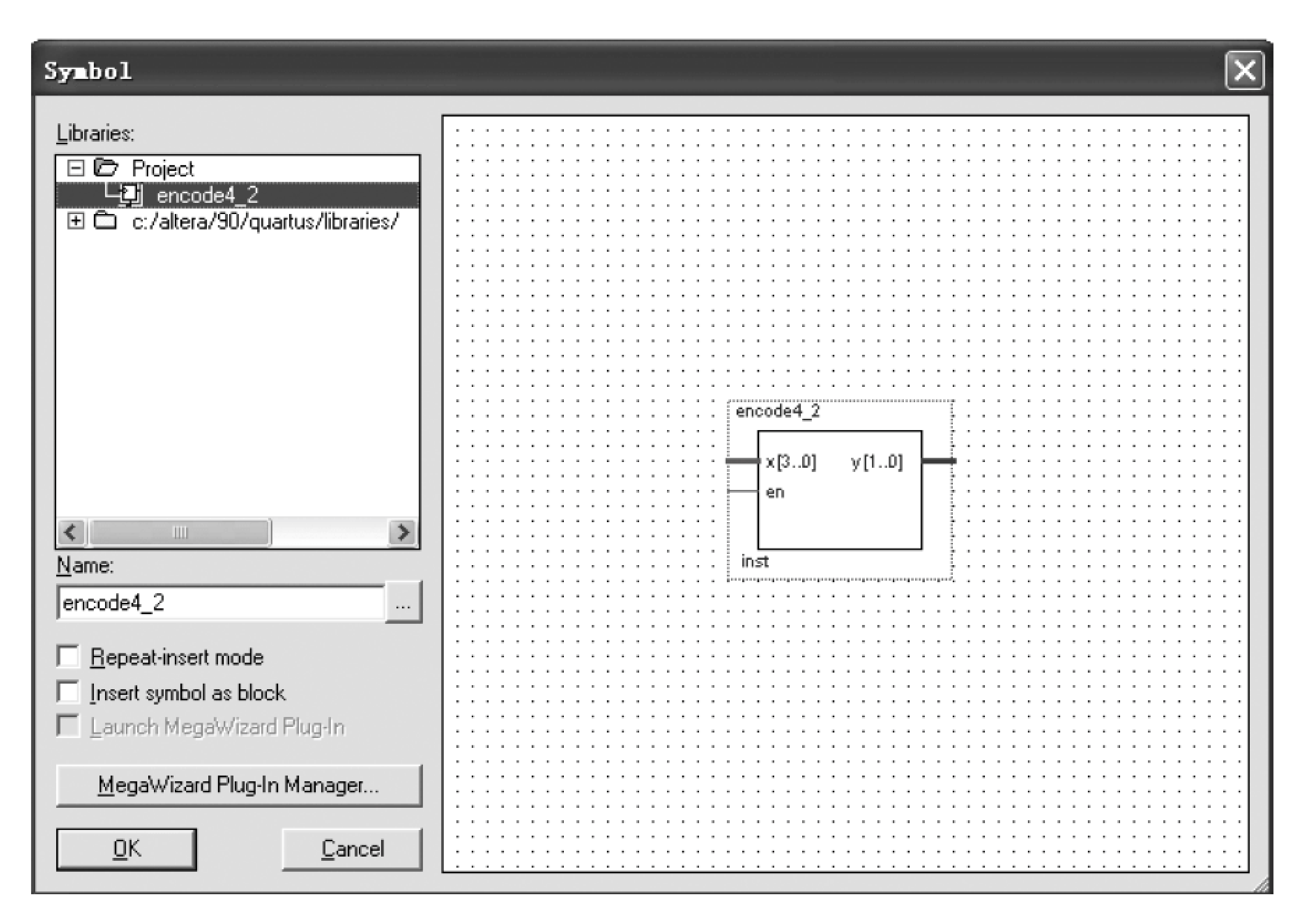


图 3-60　添加模块

为编码器添加输入输出引脚，在此要注意，在一个输入输出端口上添加多个引脚的方式和 Verilog HDL 语言中的表示方式不同，如在 x 输入端添加 4 个引脚，应表示为 iSW[3..0]，如图 3-61 所示。

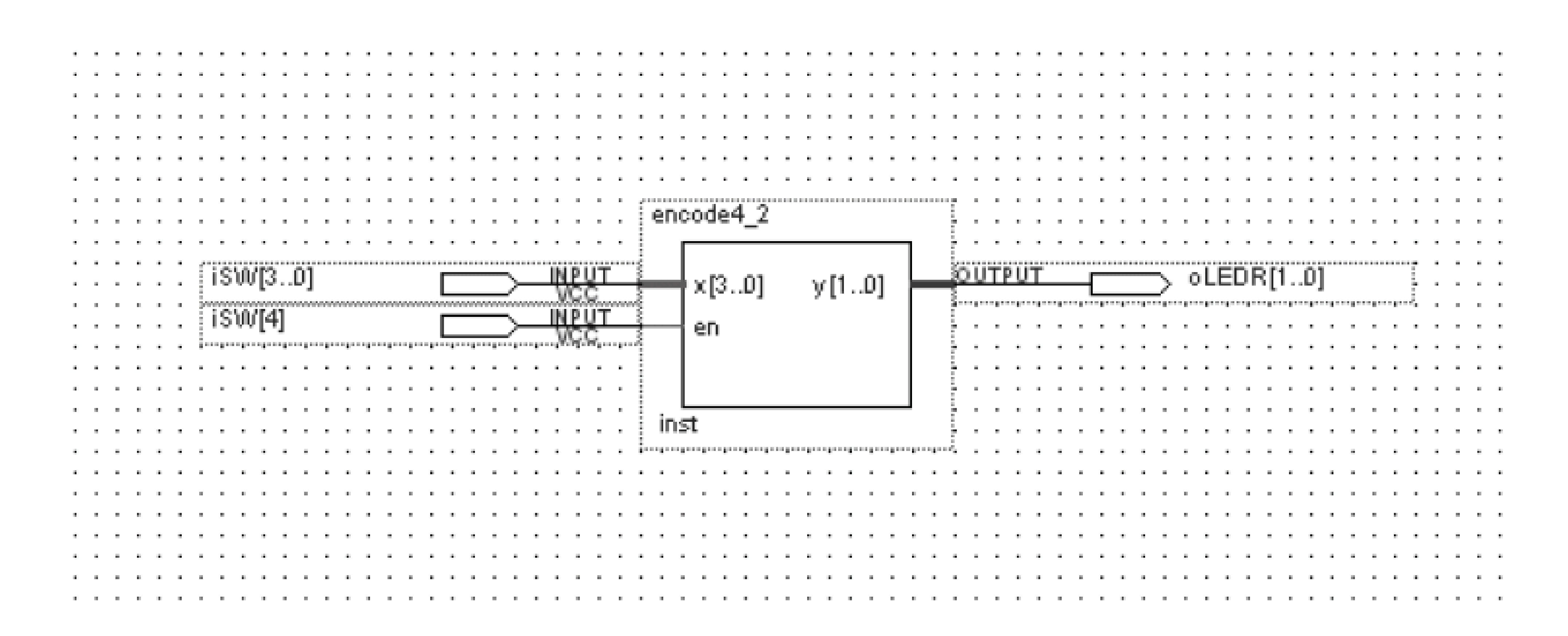


图 3-61　添加输入输出引脚

保存文件为顶层实体名.bdf(这里是 bianma4_2.bdf)，如图 3-62 所示。

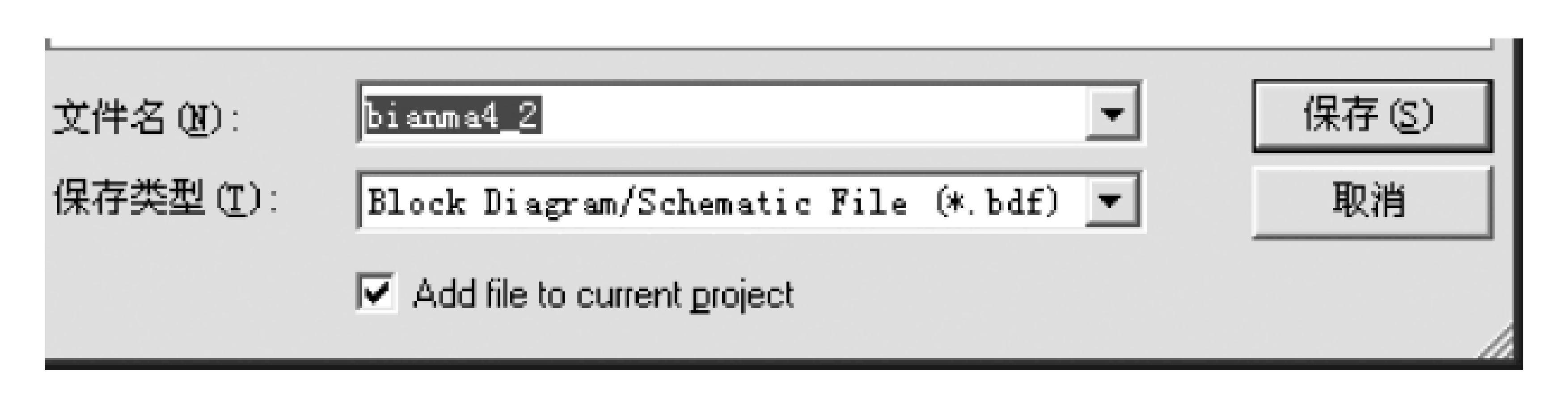


图 3-62　保存设计

此时，在项目导航的 File 栏，会出现该文件 bianma4_2.bdf，如图 3-63 所示。

将此文件设置为顶层实体文件，如图 3-64 所示。

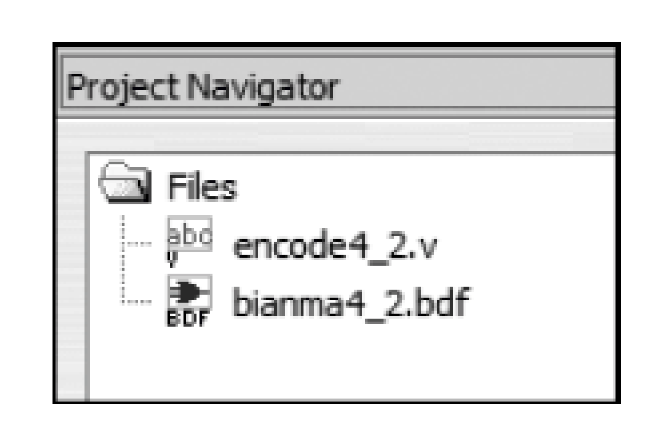


图 3-63　项目导航

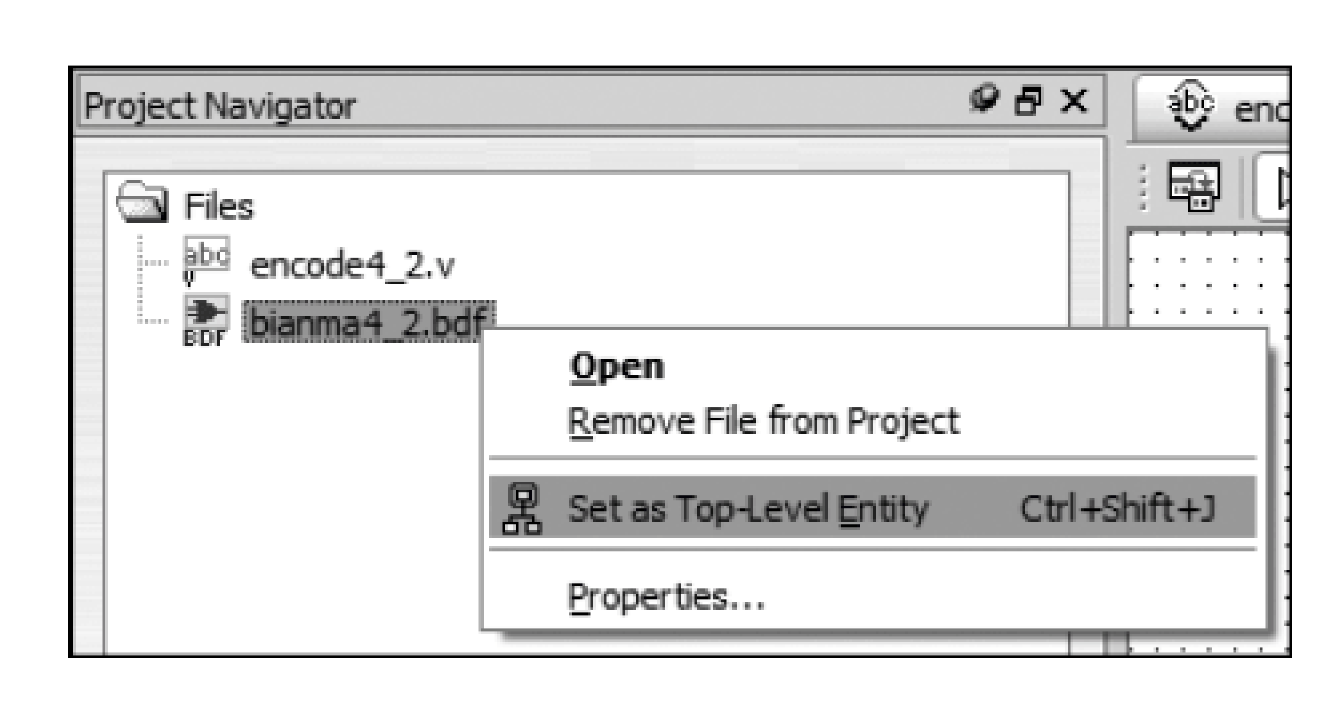


图 3-64　将文件设置为顶层实体

配置引脚，编译工程，检查错误。

仿真：如果需要，重新产生测试文件，文件默认保存为“顶层实体名. vt”，这里是“bianma4_2. vt”，如图 3-65 所示。

此时进行仿真，要将新建的“顶层实体名. vt”设为仿真文件，如图 3-66 所示。

图 3-65　生成了新的测试文件

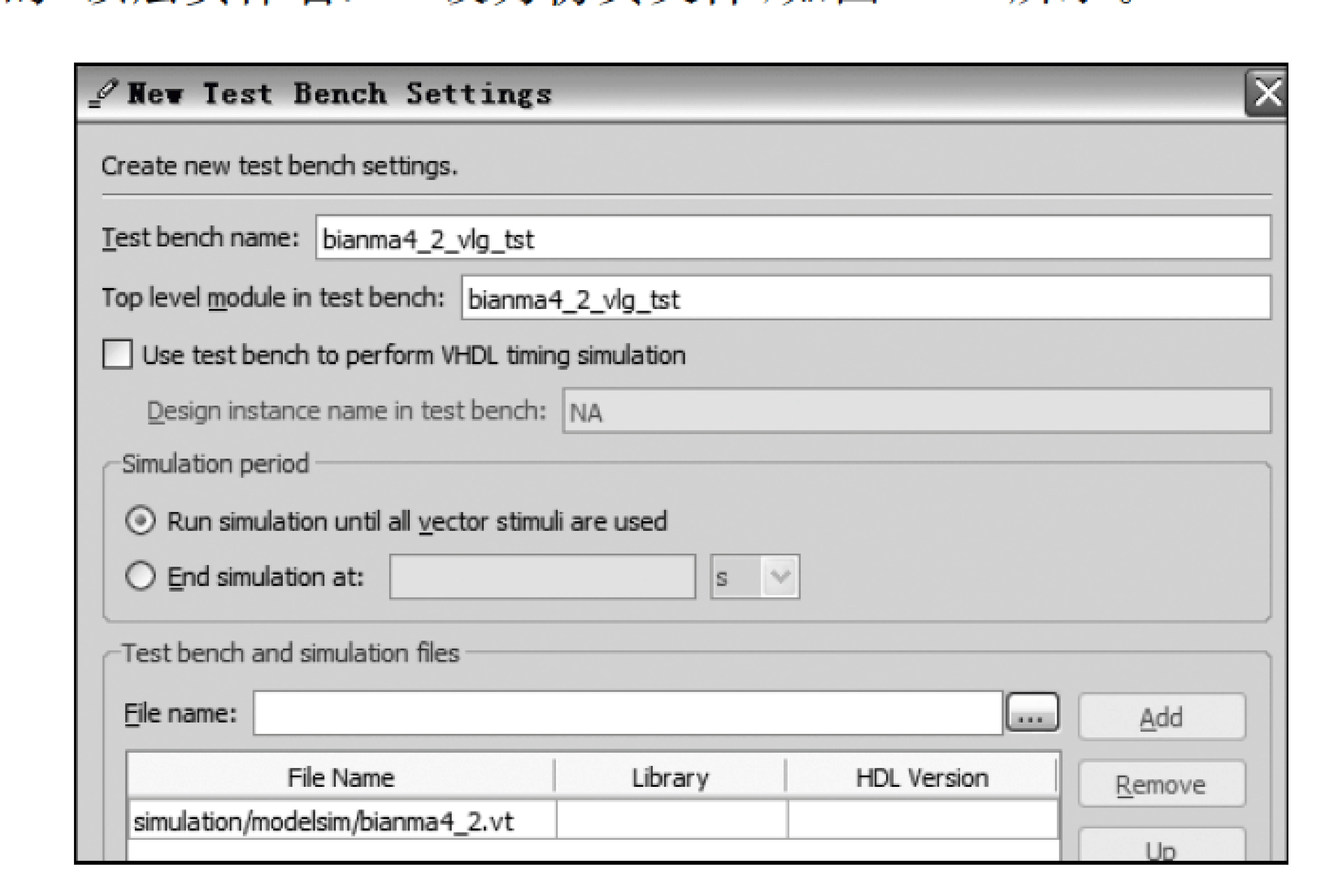


图 3-66　重新设置仿真条件

仿真正确后，将设计下载到 FPGA 中，验证设计的正确性。

3.3.2　实验内容

3.3.2.1　优先编码器

在数字系统中常常会遇到这样的情况：有几个部件同时发出服务请求，而在同一个时刻只能给一个部件发出允许操作信号，处理一个部件的请求。微处理器中的中断请求就是这样的情况。这种情况下，就要求系统能够根据任务的轻重缓急，规定好这些部件允许操作的先后顺序，即为输入信号设定优先级，当有多个输入信号有效时，编码器产生最高优先级的输入端编号，这样的编码器称为优先编码器。

74LS148 是商用 MSI 8 输入优先编码器，其逻辑符号如图 3-67 所示。

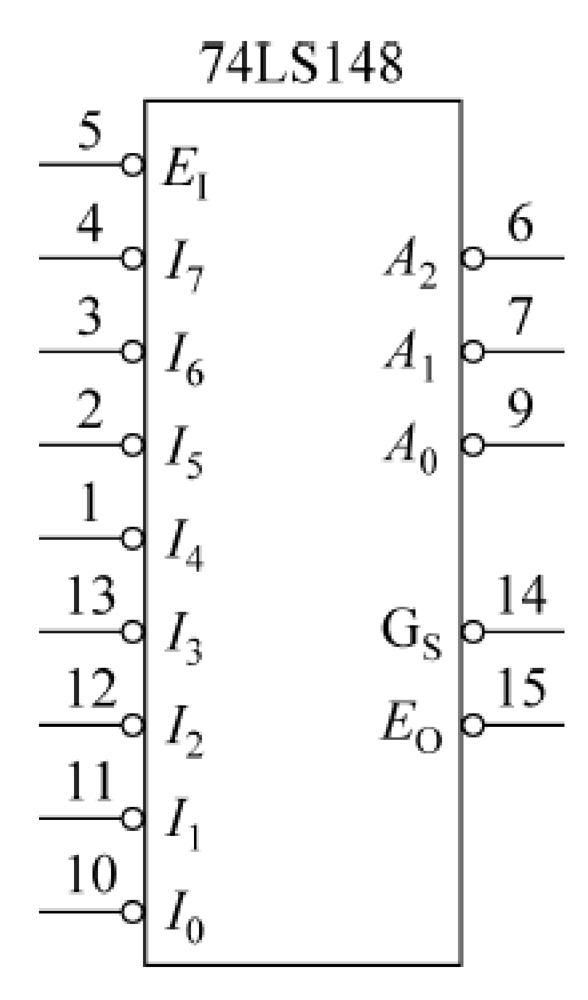


图 3-67　74LS148 输入优先级编码器的逻辑符号

74LS148 的真值表如表 3-4 所示。

表 3-4 74LS148 输入优先级编码器的真值表

E_I_L	I_0_L	I_1_L	I_2_L	I_3_L	I_4_L	I_5_L	I_6_L	I_7_L	A_2_L	A_1_L	A_0_L	G_S_L	E_O_L
1	×	×	×	×	×	×	×	×	1	1	1	1	1
0	×	×	×	×	×	×	×	0	0	0	0	0	1
0	×	×	×	×	×	×	0	1	0	0	1	0	1
0	×	×	×	×	×	0	1	1	0	1	0	0	1
0	×	×	×	×	0	1	1	1	0	1	1	0	1
0	×	×	×	0	1	1	1	1	1	0	0	0	1
0	×	×	0	1	1	1	1	1	1	0	1	0	1
0	×	0	1	1	1	1	1	1	1	1	0	0	1
0	0	1	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0

其中 E_I_L 是使能输入端，低电平有效。I_7_L～I_0_L 是优先选择器的输入端，低电平有效，I_7_L 优先级最高，I_0_L 优先级最低。G_S_L 是输出有效端，低电平有效，当 74LS148 被使能，且有一个以上的输入请求有效时，则 G_S_L 输出有效，为低电平。E_O_L 信号是用于级联时的使能输出端，级联时，它接到另一块处理较低优先级请求芯片的 E_I_L 输入；如果 E_I_L 有效，但没有有效的请求输入，则 E_O_L 有效；因此，优先级别较低的 74LS148 可以被使能。

* 设一个单片机有 I_{15}～I_0 共 16 个不同的中断请求信号，其下标码越大，优先级别越高。O_3～O_0 为中断请求信号的编码输出，输入和输出均为低电平有效。E_I 为允许输入端，G_S 为允许输出端，E_O 为编码群输出端。

请根据 74LS148 的功能，先设计一个 74LS148 优先选择器，再利用两片 74LS148 实现 16 输入的优先编码器，利用 RTL Viewer 查看你设计的电路，并利用 DE2-70 平台验证电路的正确性。

3.3.2.2 密码电路

有时候，我们和某个朋友通信时并不希望不相干的人看懂，那我们就可以采用自己专用的密码系统和朋友进行通信。

请自己设计一个编码器，将自己要发送的信息进行编码，发送给朋友(输出)。再设计一个译码器，将收到的密码(编码器的输出)译码读出，还原出真实的信息。

3.4 三态缓冲器和多路复用器

逻辑门的输出除了有高、低两种状态以外，还可能有第三种状态——高阻态，这种逻辑门就是三态门。三态门处于高阻态时，其电阻值很大，相当于该门和它所连接的电路处于断开状态。三态电路是一种重要的接口电路。了解三态门的工作和设计可以进一步理

解计算机总线工作过程。

本实验的目的是了解三态缓冲器的工作原理，学习三态缓冲器的设计，并了解三态缓冲器在计算机总线上的应用。

3.4.1 一位三态缓冲器

三态门在计算机的设计中有着重要作用，计算机的各个部件要通过总线连接在一起，而总线只允许同时有一个使用者使用，这个使用者使用总线时，其他连接在总线上的器件要处于断开状态。因此，三态门是总线连接的最好解决方案。总线上连接数个器件，每个器件通过选通信号选通，如果该器件没有被选通，相当于它与总线是断开的，不影响其他器件工作。

三态缓冲器在高速工作的 CPU 与慢速工作的外设之间起协调和缓冲作用，三态缓冲器除了常规缓冲器的功能外，还有一个选通输入端，用 E 表示。当 E 有效和 E 无效时有不同的输出状态，其结构示意图如图 3-68 所示。

当 E 有效时，选通，其输入信号直接由输出端输出 $y=x$(或反相后输出)。

若 E 无效时，缓冲器被阻止，无论输入什么值，输出端总是高阻态，用 Z 表示。高阻态能使电流降到足够低，以至于像缓冲器的输出没有与任何电路相连一样。

三态缓冲器的 Verilog HDL 模型及仿真波形如图 3-69 和图 3-70 所示，在 Modelsim 的仿真图中，高阻态以蓝色显示。

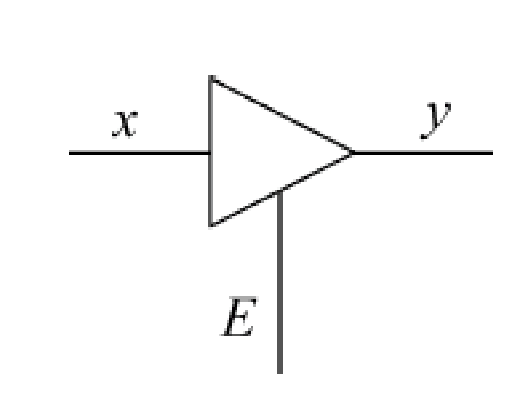


图 3-68 三态缓冲器

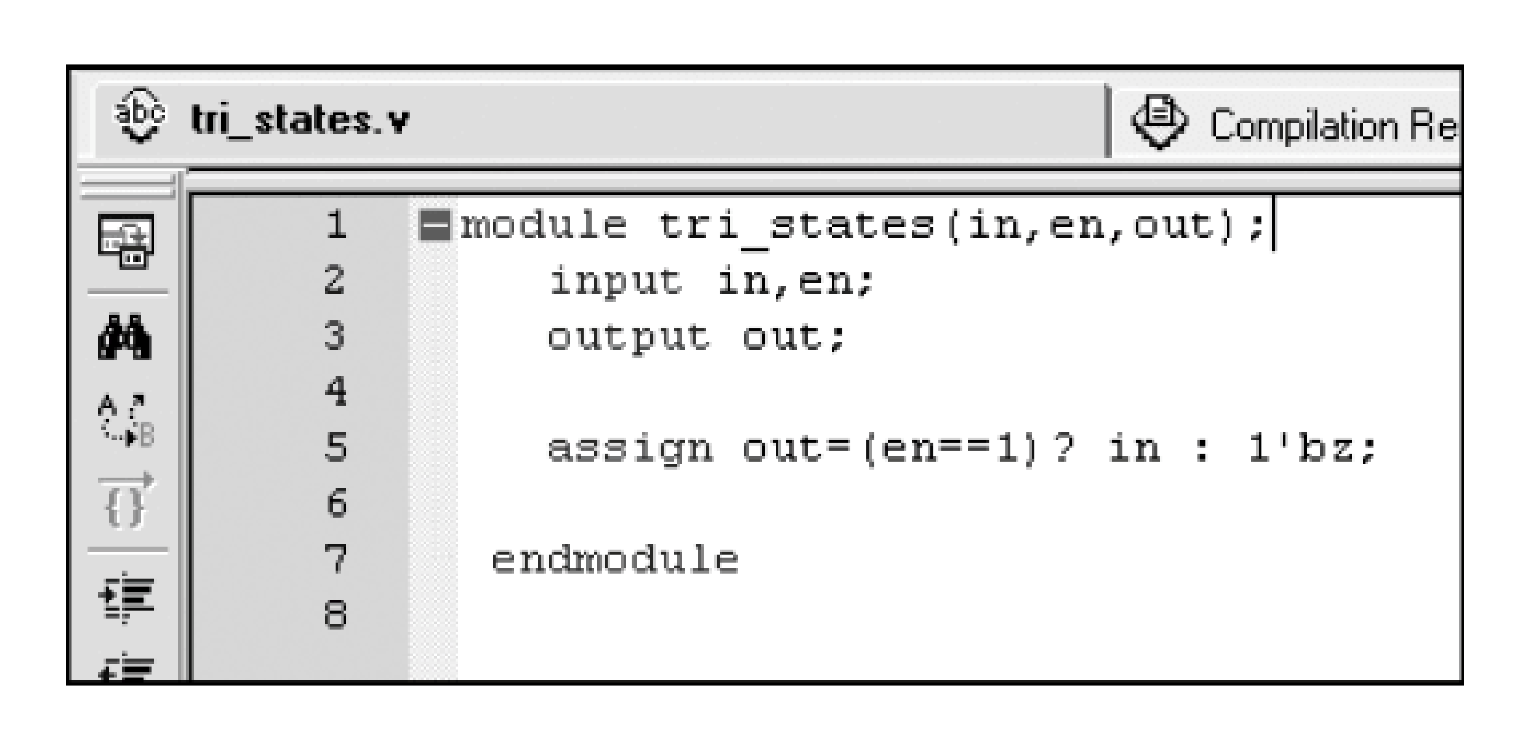


图 3-69 缓冲器设计代码



图 3-70 缓冲器仿真图

3.4.2 实验内容

3.4.2.1 8 位三态缓冲器

74LS541 是常用的 8 位三态缓冲器，74LS541 含有两个公共使能端，G_1_L 和 G_2_L，

低电平有效。只有当两个使能端都有效时，器件的三态输出端才能有效地输出。图 3-71 是 74LS541 的逻辑原理图。

请查询资料，根据 74LS541 的功能，设计一个 74LS541 三态缓冲器，采用适当的方式，验证你设计的正确性。

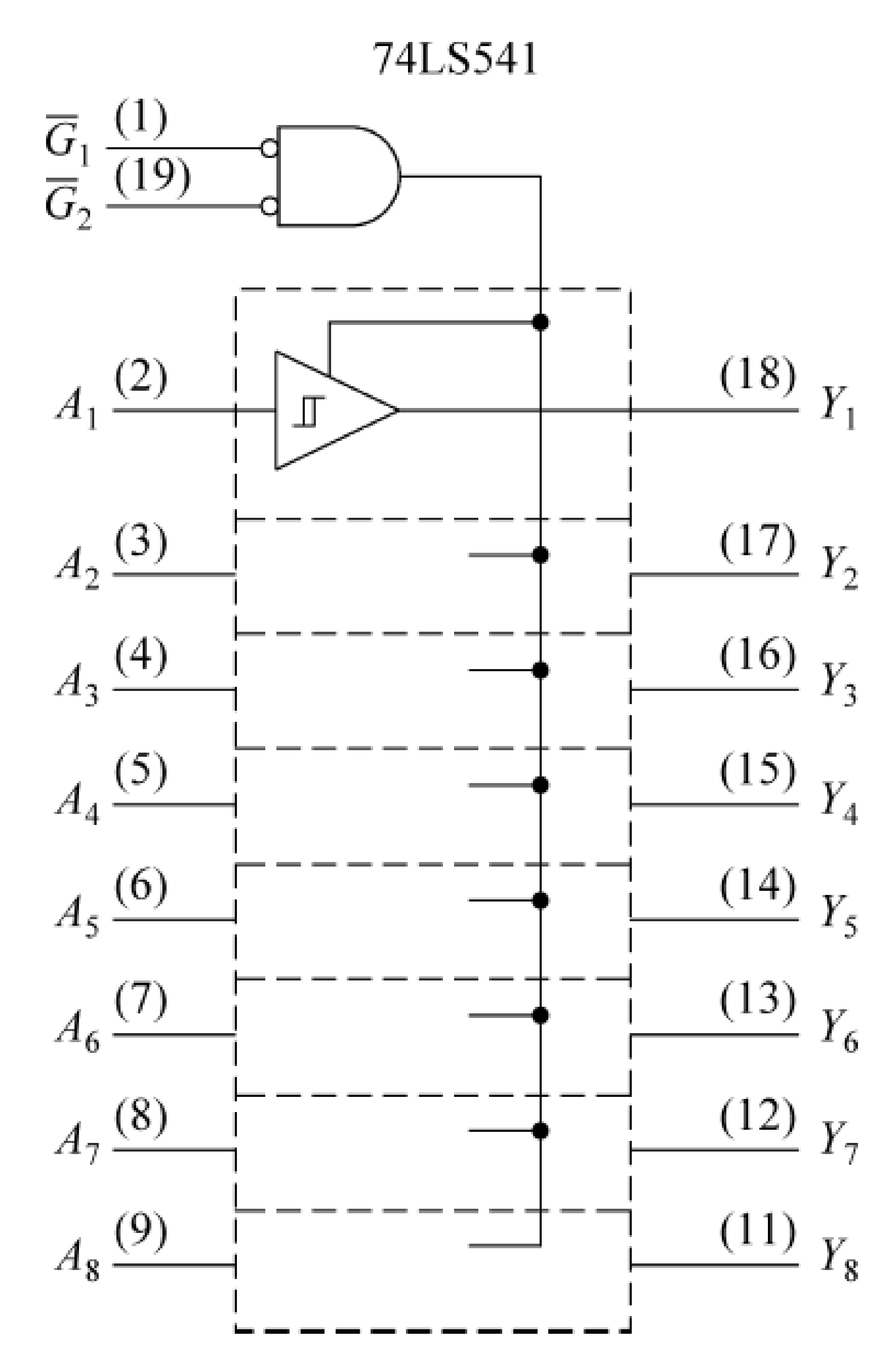


图 3-71　74LS541 逻辑原理图

3.4.2.2　多路选择器、多路分配器和总线

多路选择器用于选择多个输入中的一个输入，将其送到同一条总线上。在计算机中，多个器件同时连接在一条总线上，多路分配器就是选择这些器件中的一个，并将总线上的数据送到此器件中，而不会与其他器件产生数据冲突。

图 3-72 是一个驱动总线的多路选择器和接收总线的多路分配器的逻辑框图。请设计一个多路选择器，一个三态缓冲器和一个译码器，完成图 3-72 中的设计。根据硬件资源选择总线宽度，可以是 1 位、2 位、4 位甚至 8 位，分配控制端采用译码器来控制，可参考图 3-73 的方案，顶层实体采用电路原理图方式设计，灵活应用硬件资源，验证设计的正确性。

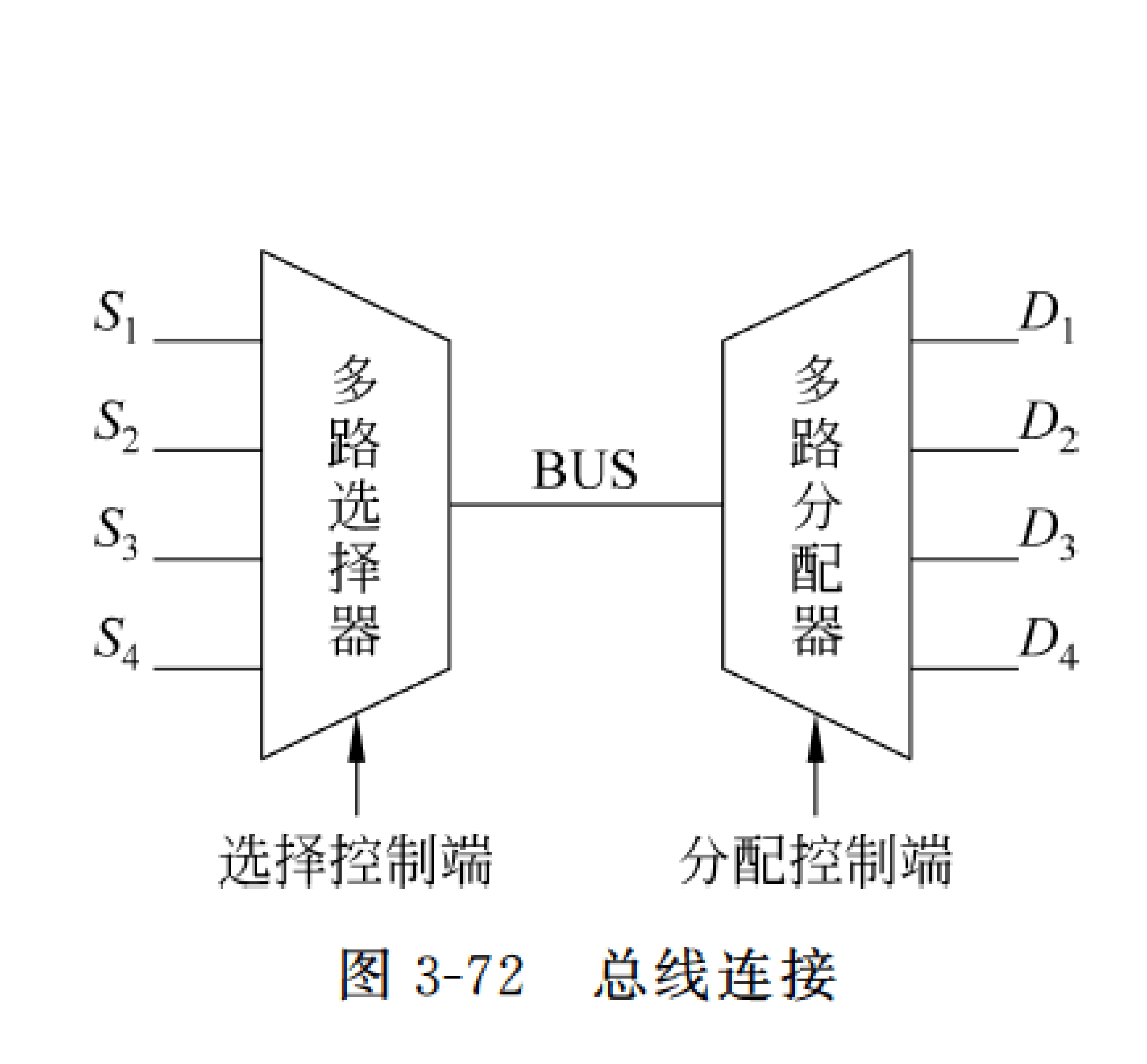


图 3-72　总线连接

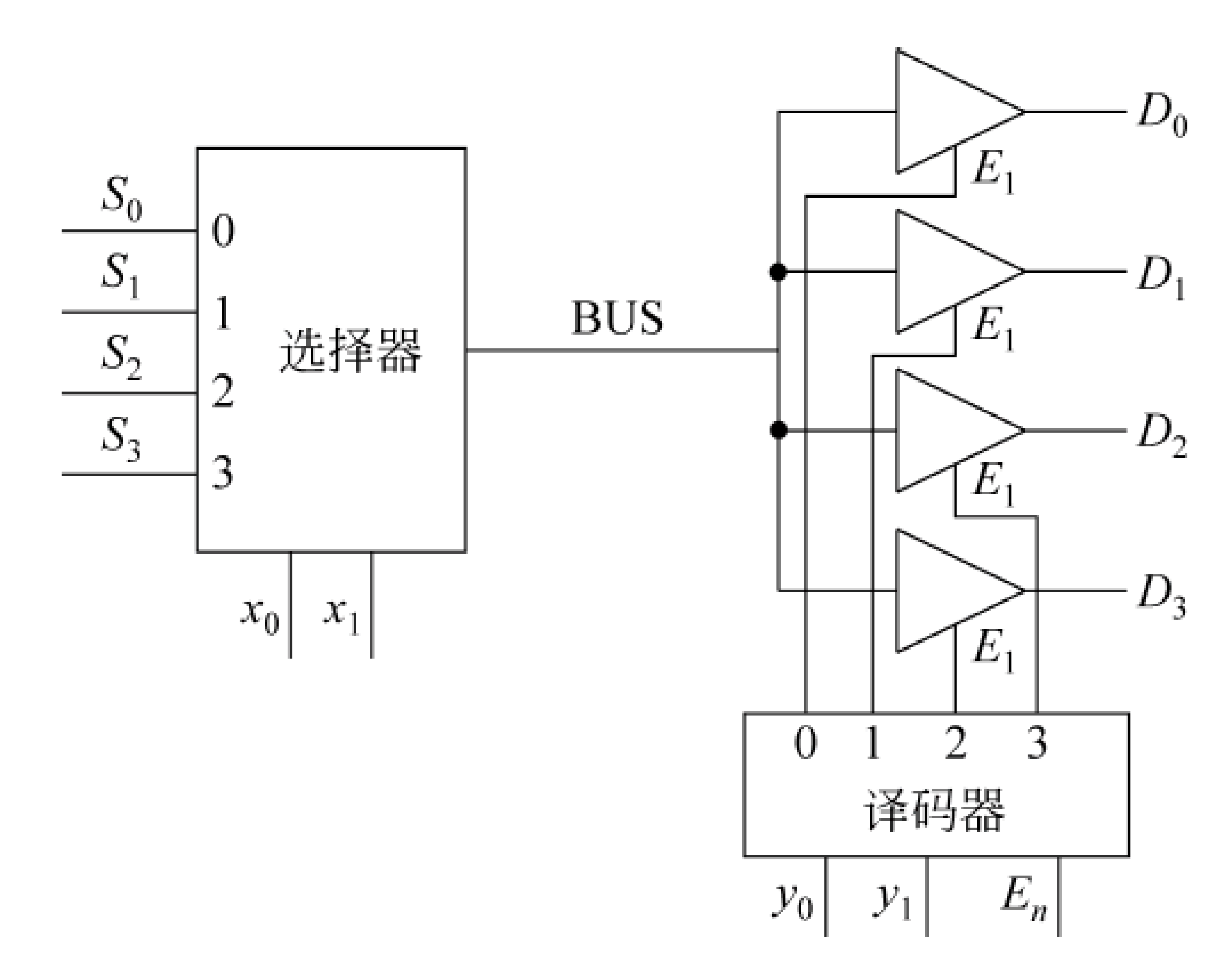


图 3-73　总线实现图

3.4.2.3*　双向三态缓冲器

图 3-74 是一个双向三态门，请查阅相关资料，了解双向三态缓冲器，并设计一个电路，举例说明双向三态缓冲器的用途。

* 本小节为选学内容。

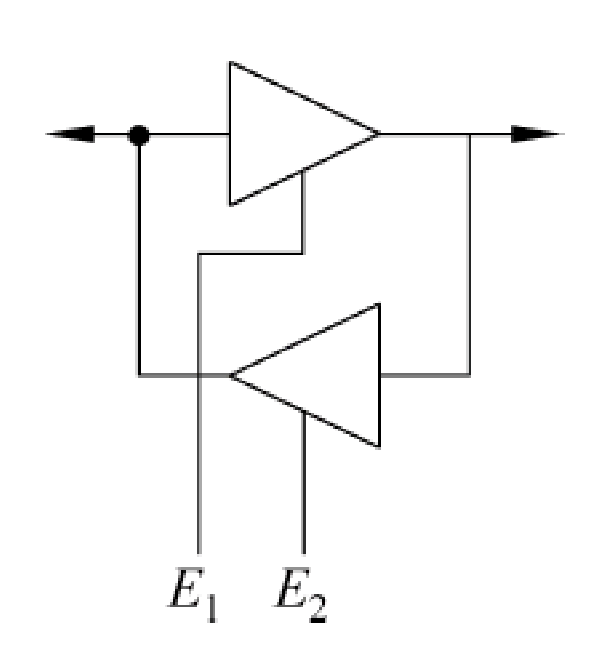


图 3-74　双向三态缓冲器

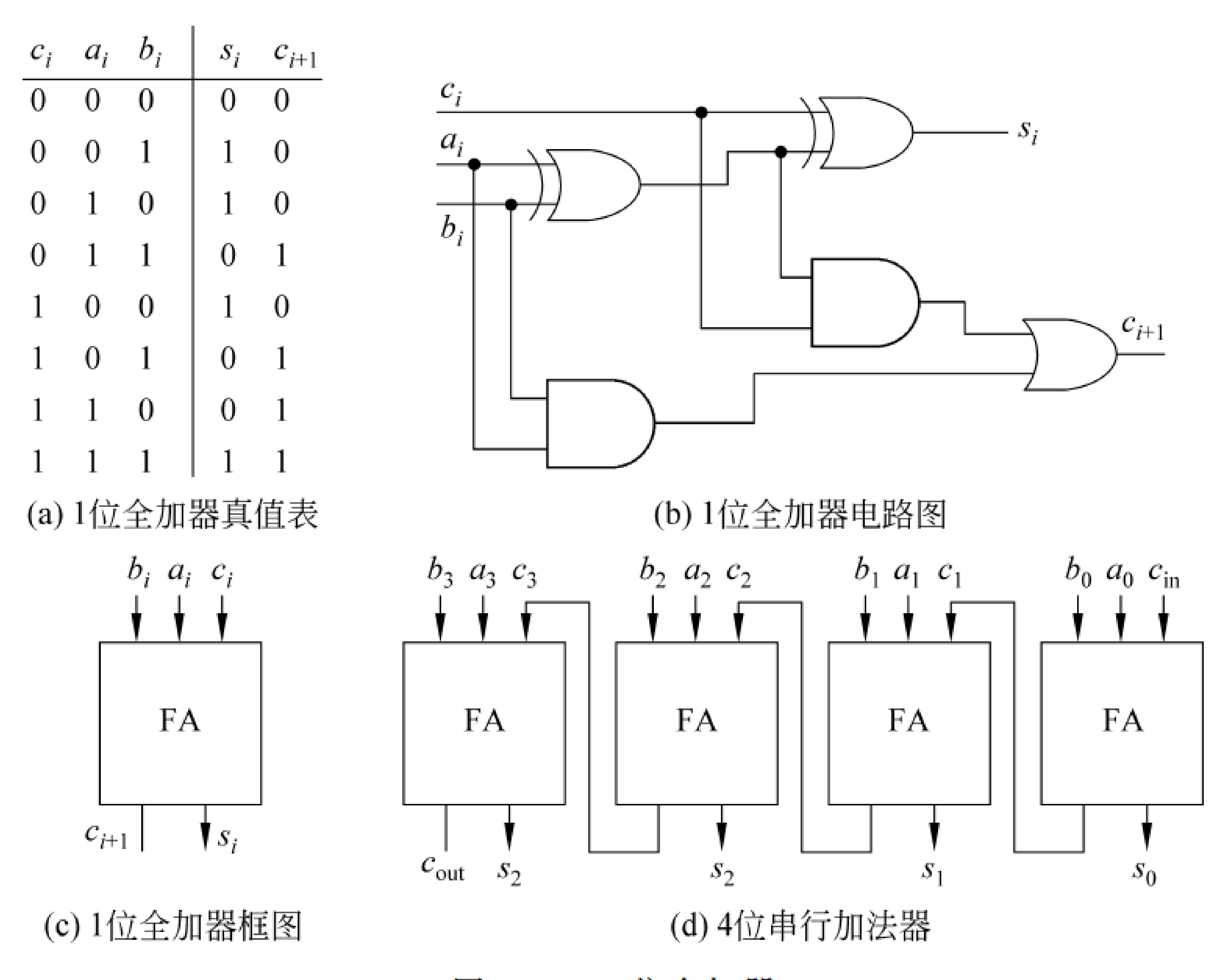
3.5　简单加法器和乘法器

加法是数字系统中最常执行的运算，加法器是 ALU 的核心部件。减法可以看作是被减数与负的减数的补码相加，因此，可以用加法器同时实现加法和减法两种运算。乘法也可以利用移位相加的算法来实现，因此加法器可以说是计算机中最“繁忙”的部件了。

本实验的目的是复习 1 位全加器的原理，学习用门级原语、算术赋值语句和利用 Altera 公司提供的参数化功能模块完成加法器的设计，比较硬件电路的各种设计方式。

3.5.1　1 位加法器

多位加法器可以由 1 位加法器级联而成，图 3-75(a)是 1 位全加器真值表，输入为 a_i、b_i 和 c_i，输出为 s_i 和 c_{i+1}；图 3-75(b)是 1 位加法器电路图；图 3-75(c)是 1 位全加器的框图；图 3-75(d)是 4 位行波进位加法器框图。输入为 $a(a_0 \sim a_3)$、$b(b_0 \sim b_3)$和 c_{in}，输出为 $s(s_0 \sim s_3)$和 c_{out}。



c_i	a_i	b_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a) 1位全加器真值表

(b) 1位全加器电路图

(c) 1位全加器框图

(d) 4位串行加法器

图 3-75　1 位全加器

1位全加器的设计相对简单,请读者根据电路图自行设计一个加法器电路,并验证其正确性。

串行加法器速度很慢,因为进位必须从最低位传至最高位。要想构建速度较快的加法器,就要利用附加逻辑,提前算出进位信息,这就是先行进位加法器的设计思想,先行进位加法有几种常用的算法,感兴趣的同学可以查找资料阅读。

3.5.2 实现一个8位加法器

Quartus Ⅱ的CAD系统有一个参数化模块库(LPM),LPM库中的每个模块都是与技术无关的,也是参数化的,因此可以以多种方式应用。例如,LPM库中有一个名为LPM_ADD_SUB的n位加法器模块。LPM_ADD_SUB的原理图如图3-76所示,这个模块的几个参数都可以通过CAD工具来设置。

新建一个Quartus工程,取名为"adder",如图3-77所示。

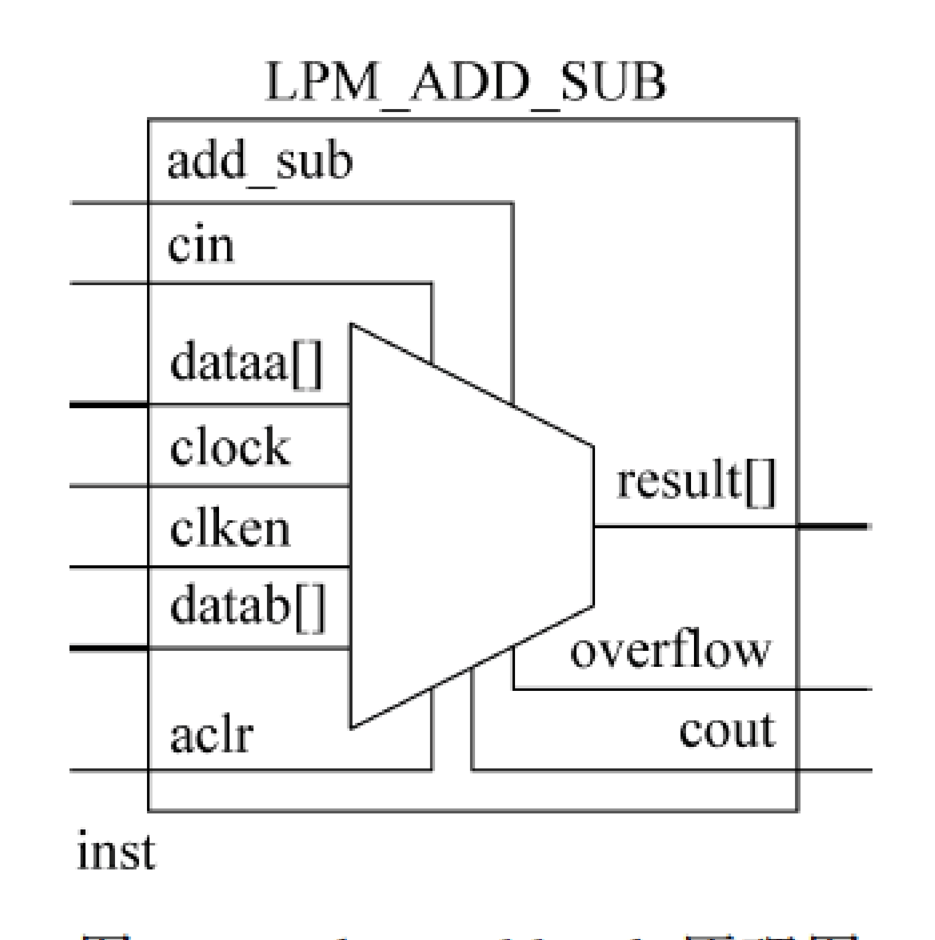


图3-76 lpm-add-sub原理图

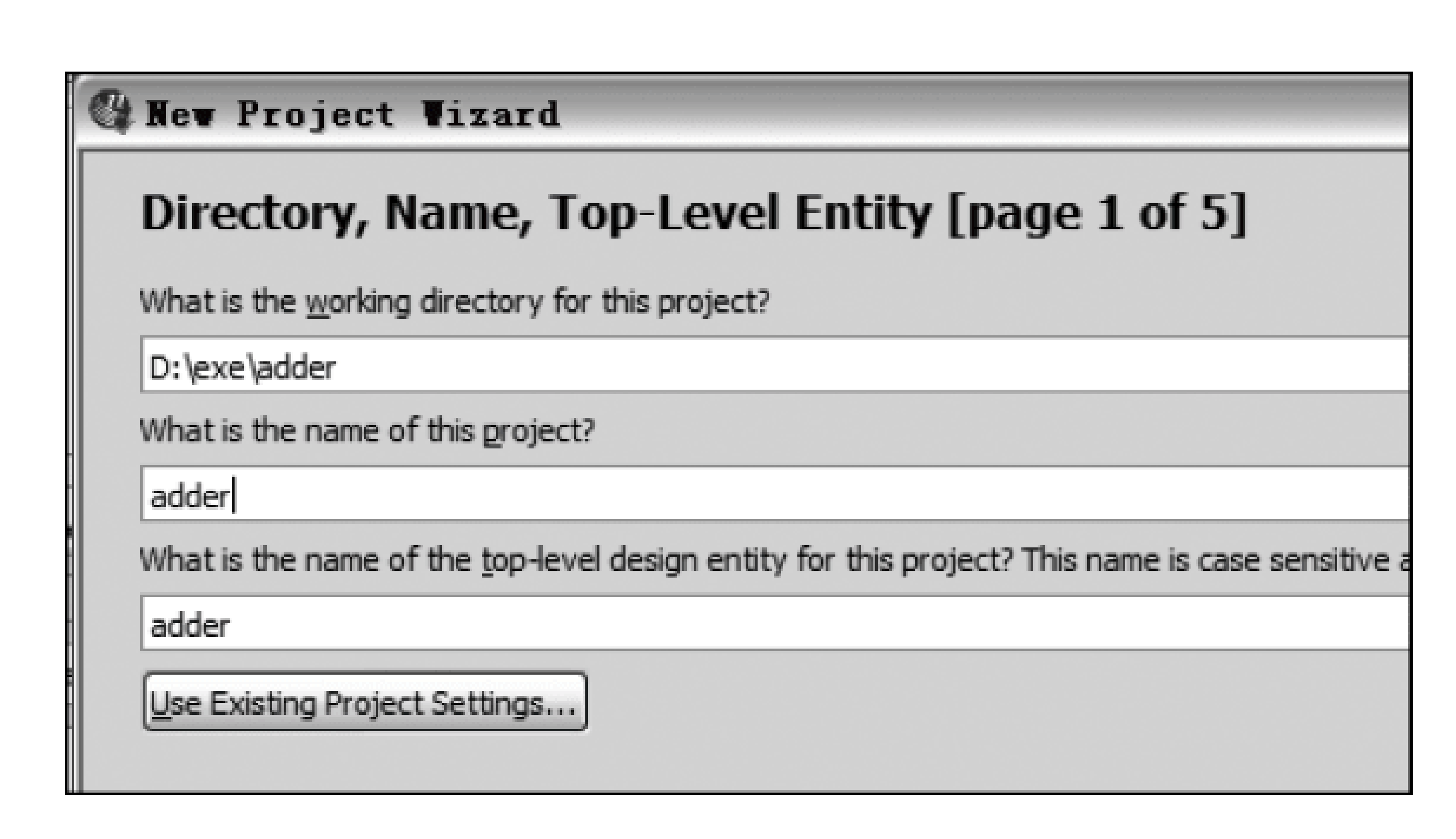


图3-77 新建工程

然后用MegaFunction创建加法器模块,首先依次单击菜单中的"Tools→MegaWizard Plug-In Manager"选项,如图3-78所示。

打开后的界面如图3-79所示。选择"Create a new custom megafunction variation",新建一个模块。选好后单击"Next"按钮。

接下来会出现如图3-80所示的界面。在图的左侧选择"Installed Plug-Ins→Arithmetic→LPM_ADD_SUB"选项,并在图的右侧输出文件名一栏中为此模块起一个名字。本例中输入命名为"adderlpm"。建立模块时系统将根据用户选择,产生一个硬件描述语言文件(AHDL、VHDL或者Verilog语言),此文件可在用硬件描述语言设计电路时直接使用。完成后单击"Next"按钮。

完成后出现图3-81所示的界面。按图选择选项,根据自己的要求设置此加法器的功能:仅加、仅减或者可加可减。单击"Next"按钮进入下一页。

选择加法器的输入值是否为常量、有符号或无符号,如图3-82所示。单击"Next"按钮进入下一页。

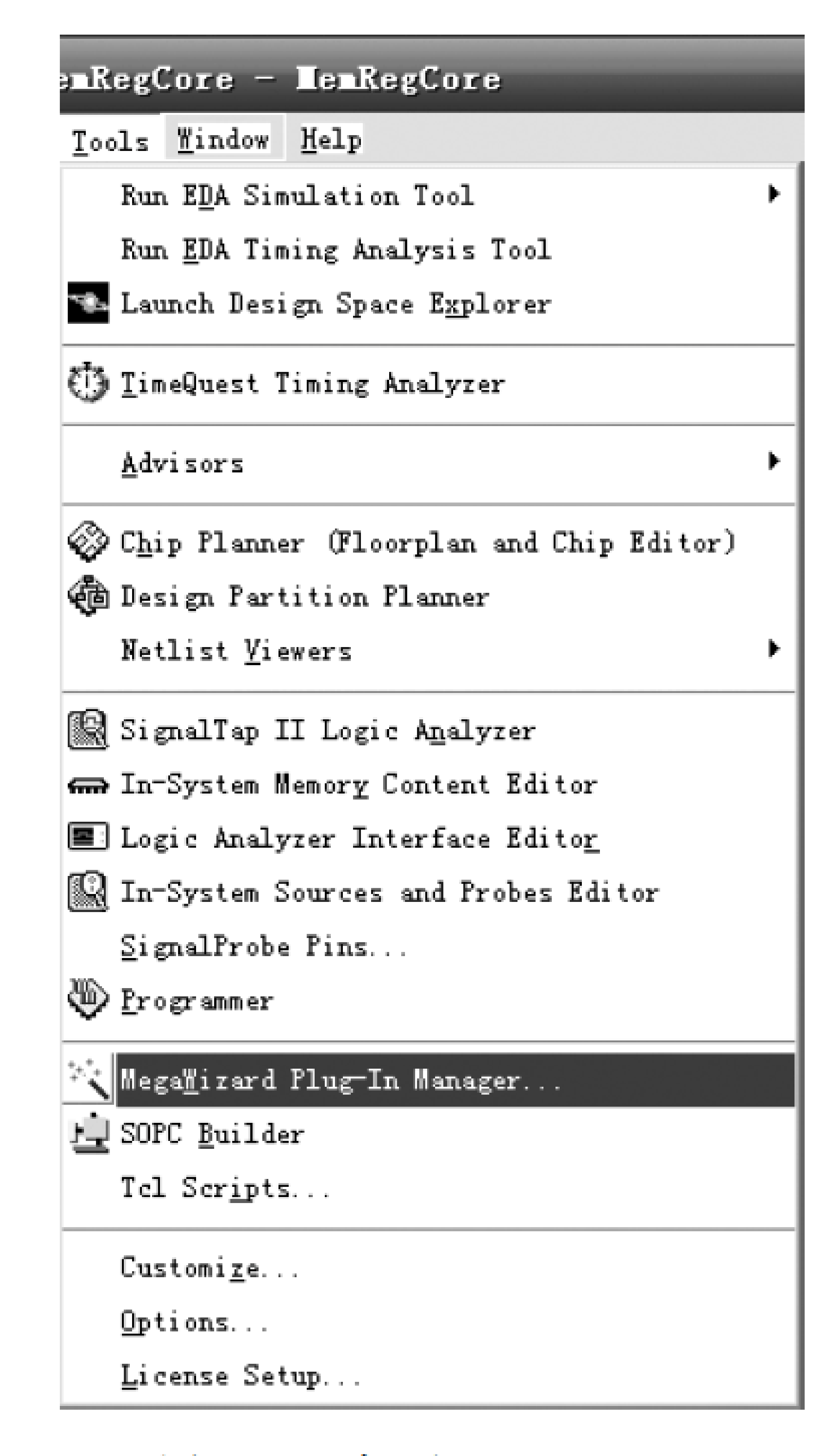


图 3-78　打开 MegaWizard

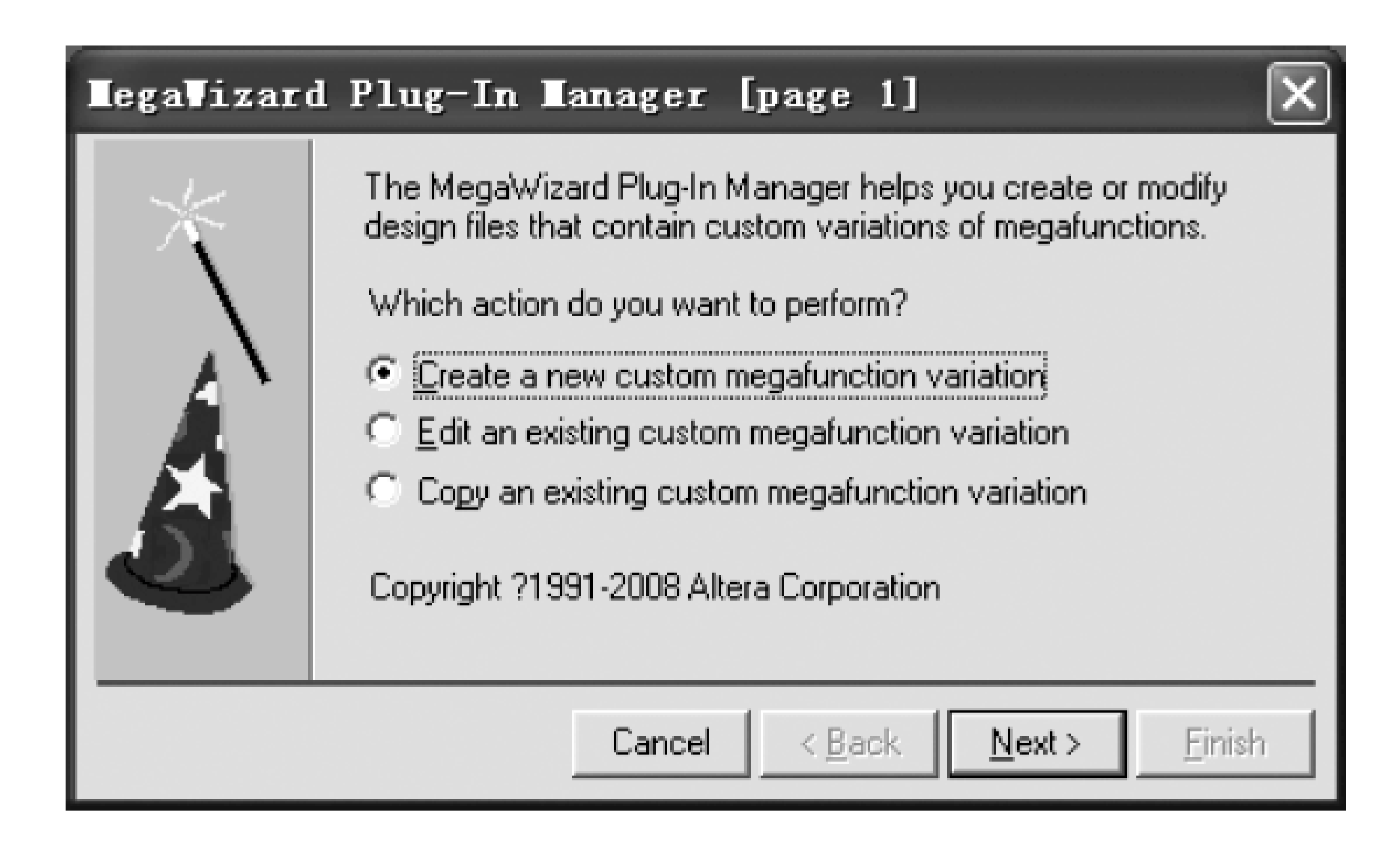


图 3-79　MegaFunction 第一步

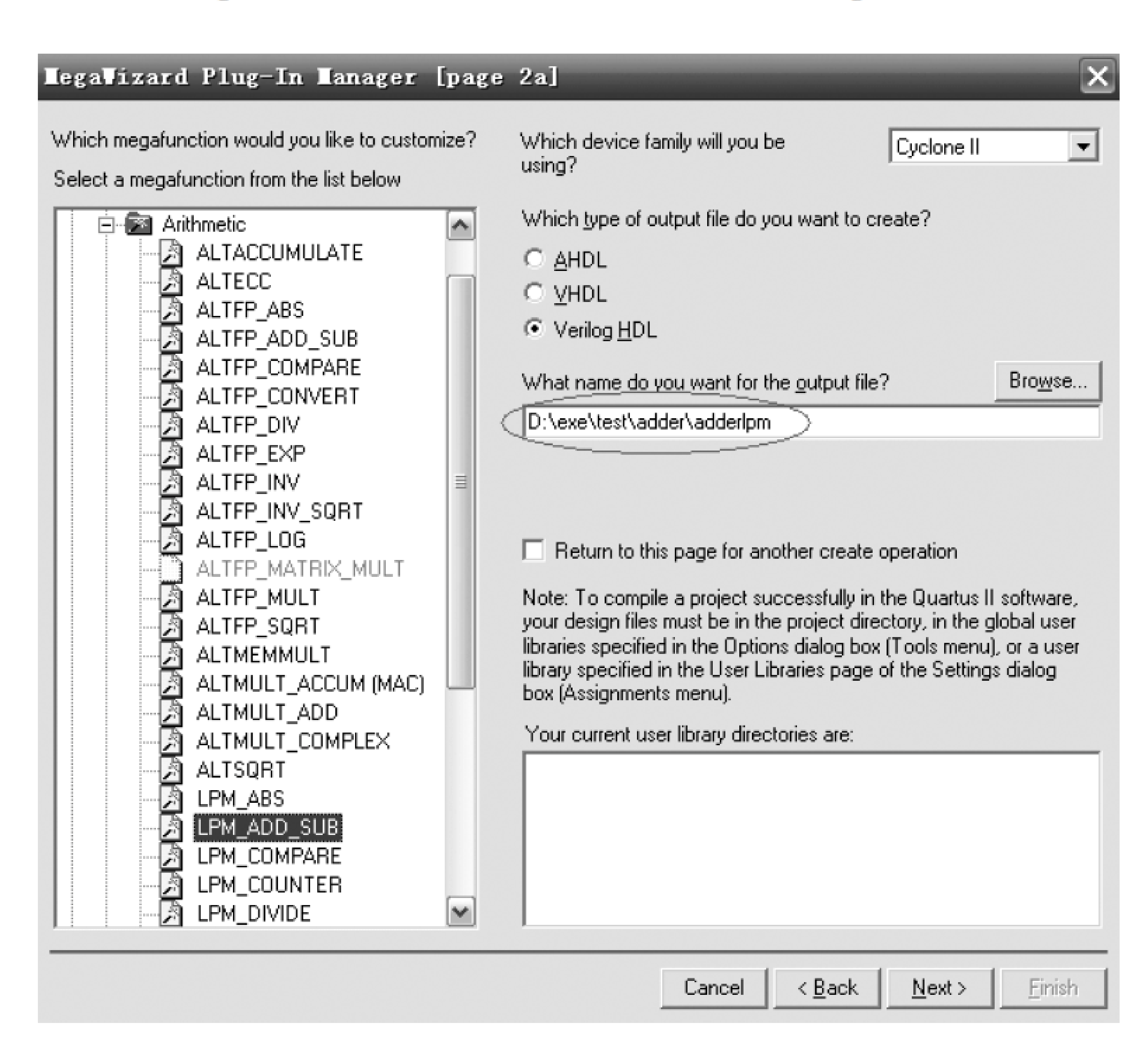


图 3-80　选择模块

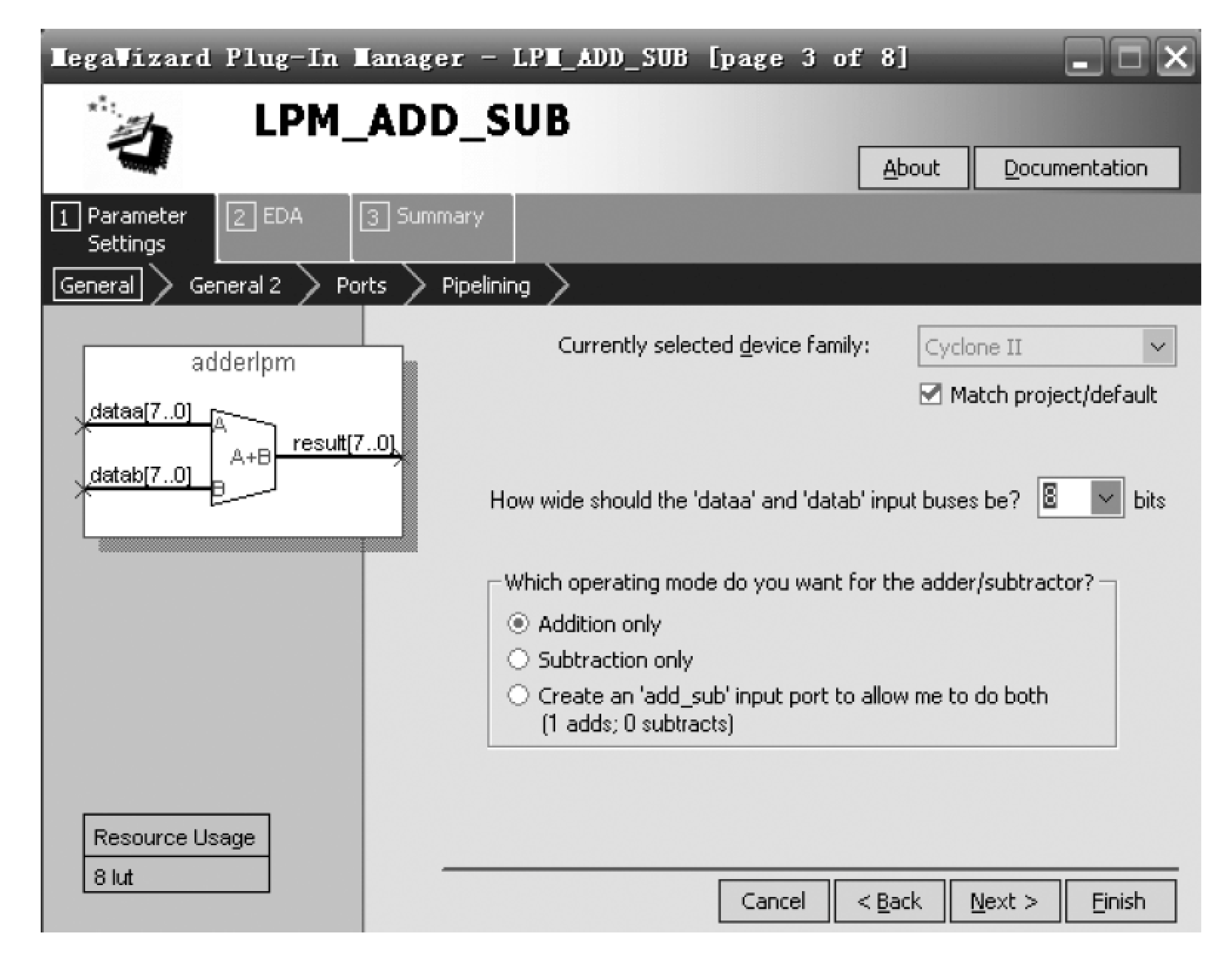


图 3-81 选择加法器类型

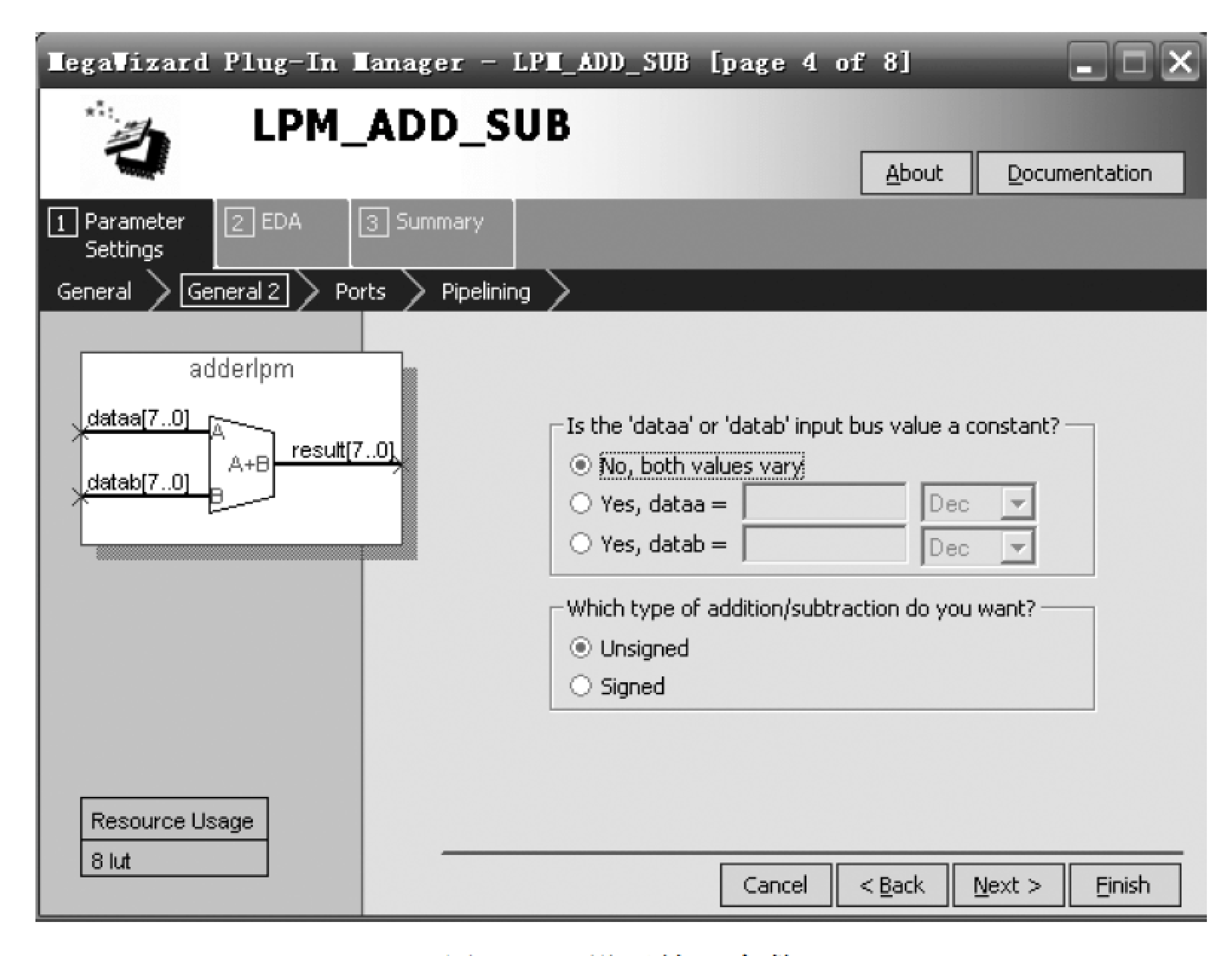


图 3-82 设置输入参数

选择加法器是否有进位输入、进位输出和溢出输出，如图 3-83 所示。单击“Next”按钮进入下一页。

图 3-83 选择进位和溢出位

选择是否用于流水操作(Pipelining)，如图 3-84 所示。单击“Next”按钮进入下一页。

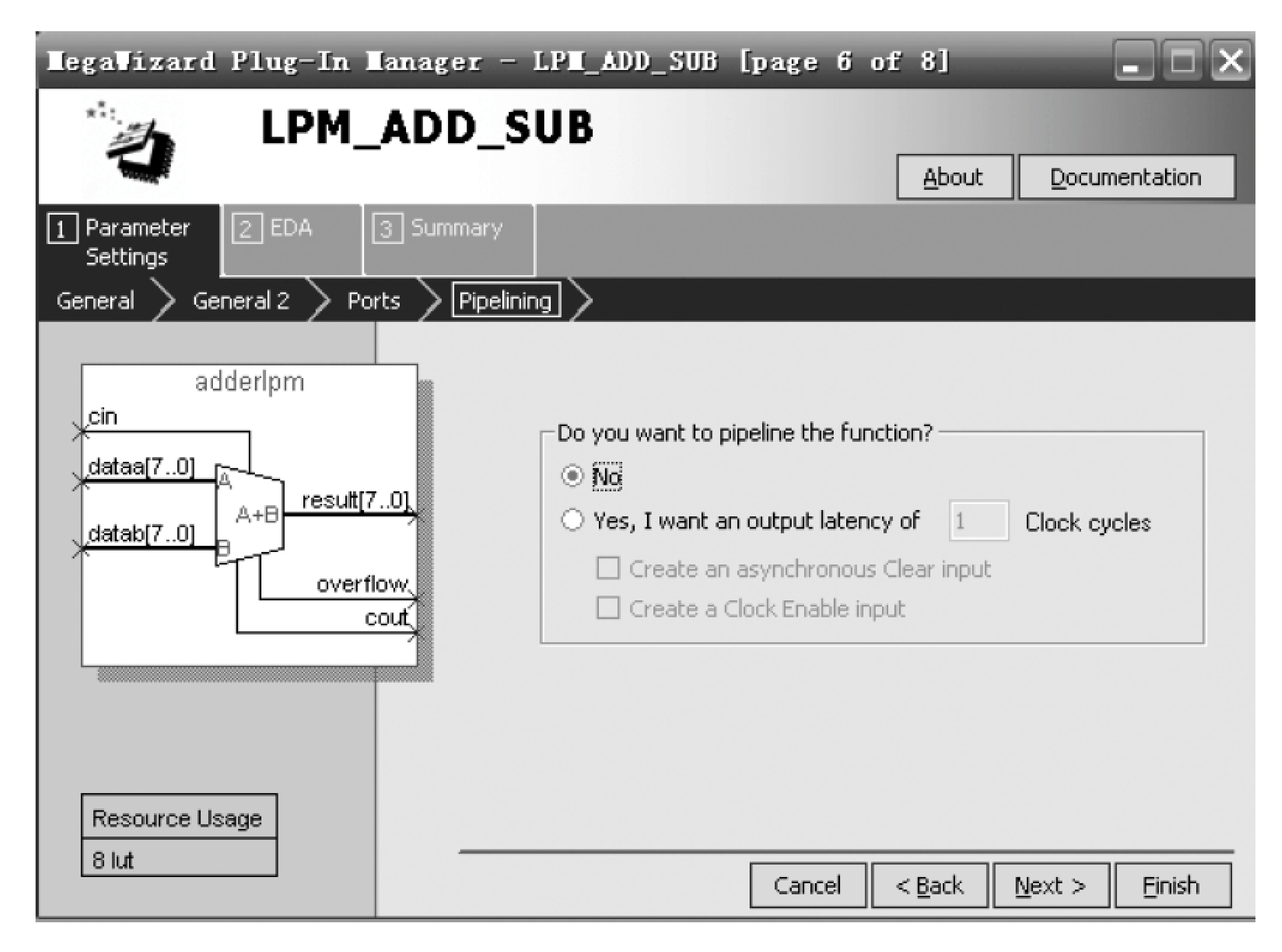


图 3-84 选择是否流水操作

单击“Next”按钮后出现图 3-85 所示界面，可以选择生成相应的仿真库，本实验采用默认操作。单击“Next”按钮，进入下一页。

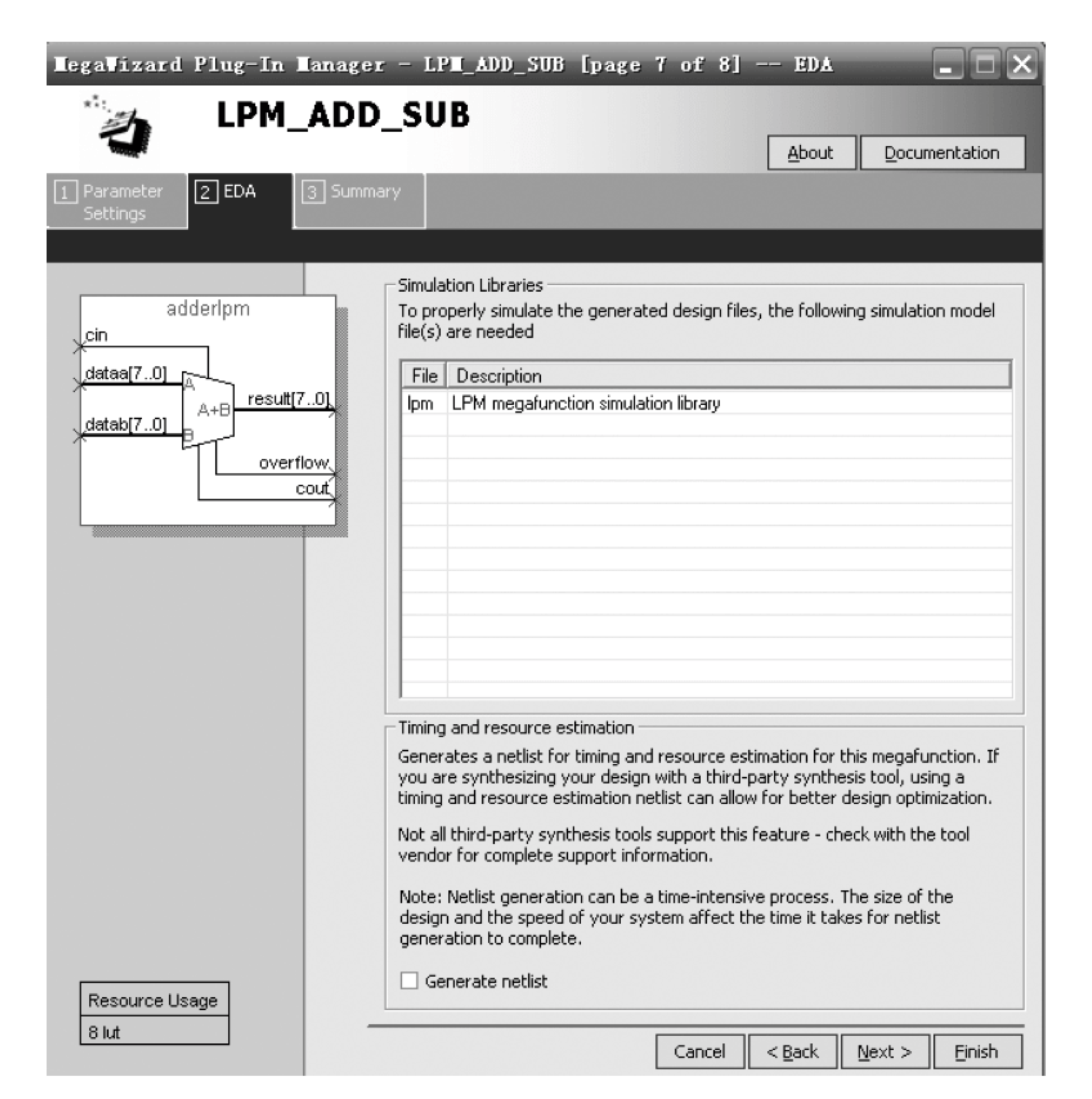


图 3-85　与仿真工具接口

完成后如图 3-86 所示。该页给出了可选的生成文件。注意选择“adderlpm. bsf”文件以生成符号文件，为符号框图输入提供源文件。系统已经默认生成“adderlpm. v”文件。完成此步后直接单击“Finish”按钮完成 MegaFunction 设计。完成后如果有对话框问你是否添加此 ip，选“NO”。

创建顶层实体描述文件。本次实验使用符号框图来完成这一步。实际上这与使用 Verilog 语言实现顶层实体完全等价。首先单击新建文件菜单选项“File→New”，选择其中的“Block Diagram/Schematic File”选项，如图 3-87 和图 3-88 所示。

加入加法器模块。在文件空白处双击鼠标左键进入选择模块对话框。在左侧选择“Project→adderlpm”选项，即刚才创建的模块，单击“OK”按钮可以选择插入，在适当位置单击左键插入，如图 3-89 和图 3-90 所示。

加入输入输出引脚。依照上一步，选择输入输出引脚，插入到符号框图中。连接并修改引脚名称。注意，如果是引脚数组，其写法为 iSW[7..0]，这与硬件描述语言中的 iSW[7:0]不同。

另外，我们同样可以用 Verilog 语言来完成设计，在工程目录下，可以找到一个名为“adderlpm. v”的文件，打开此文件，查看接口参数，此文件可以在顶层实体模块中被直接调用，如图 3-91 所示。

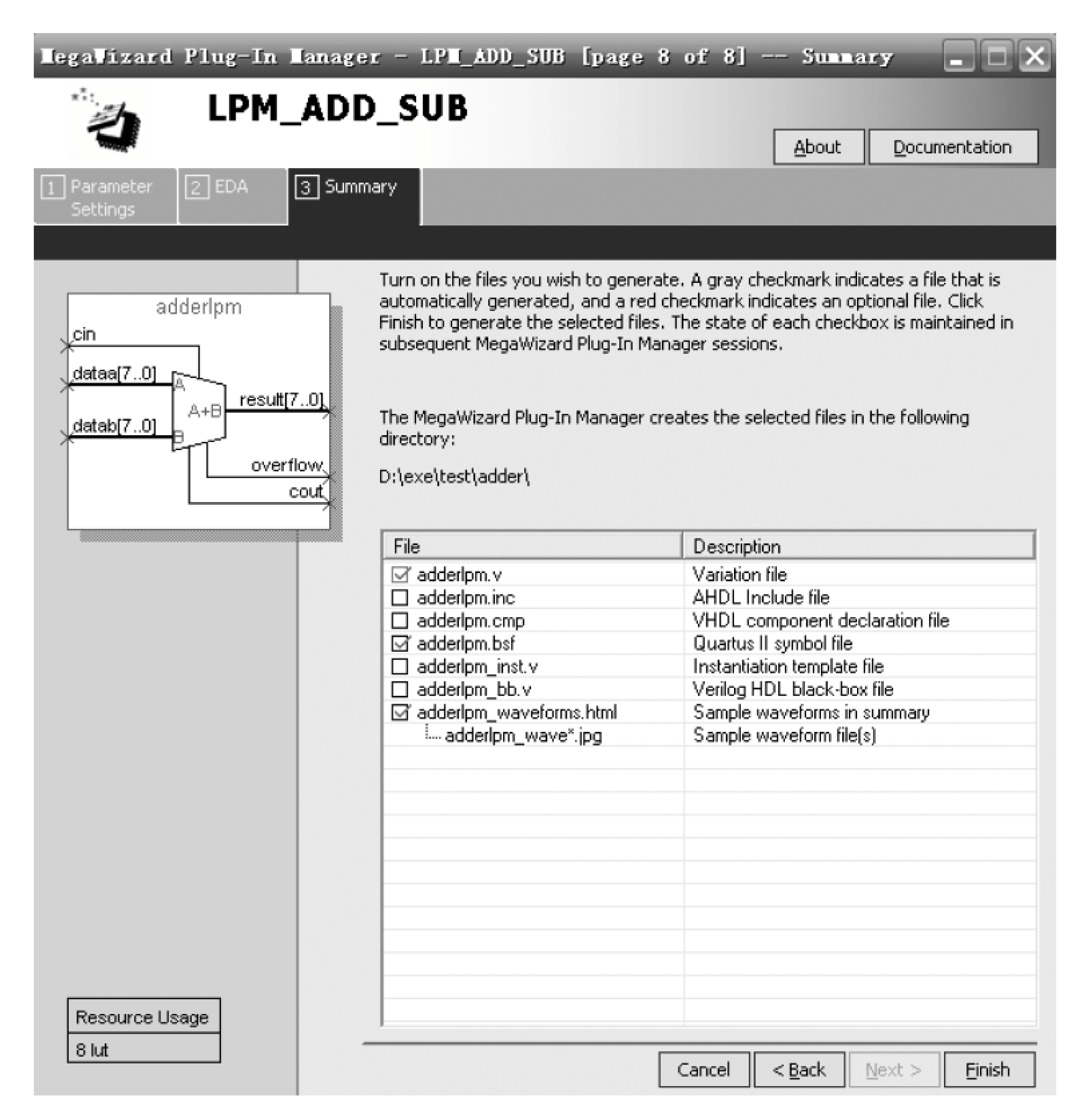


图 3-86　选择生成文件

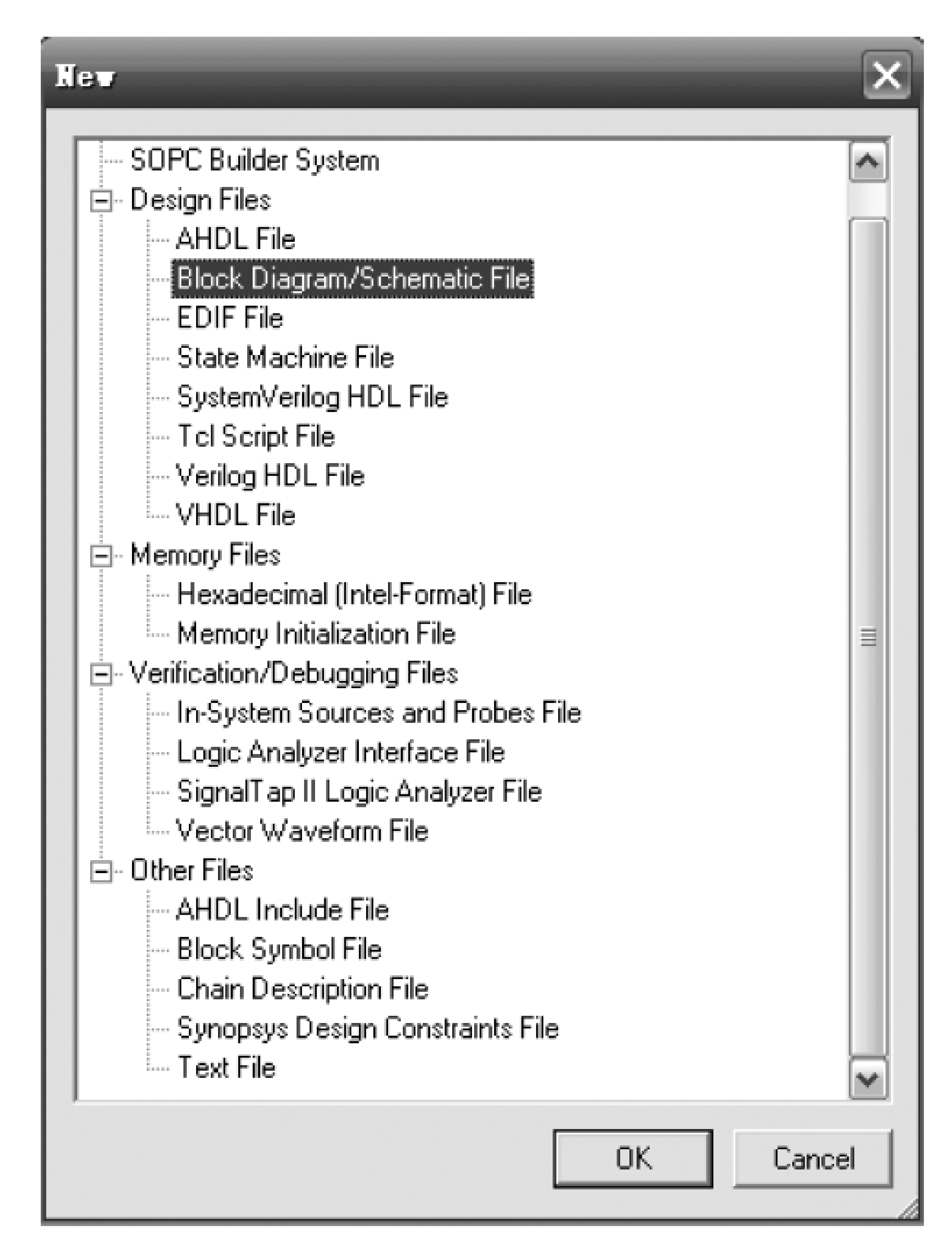


图 3-87　创建顶层实体文件

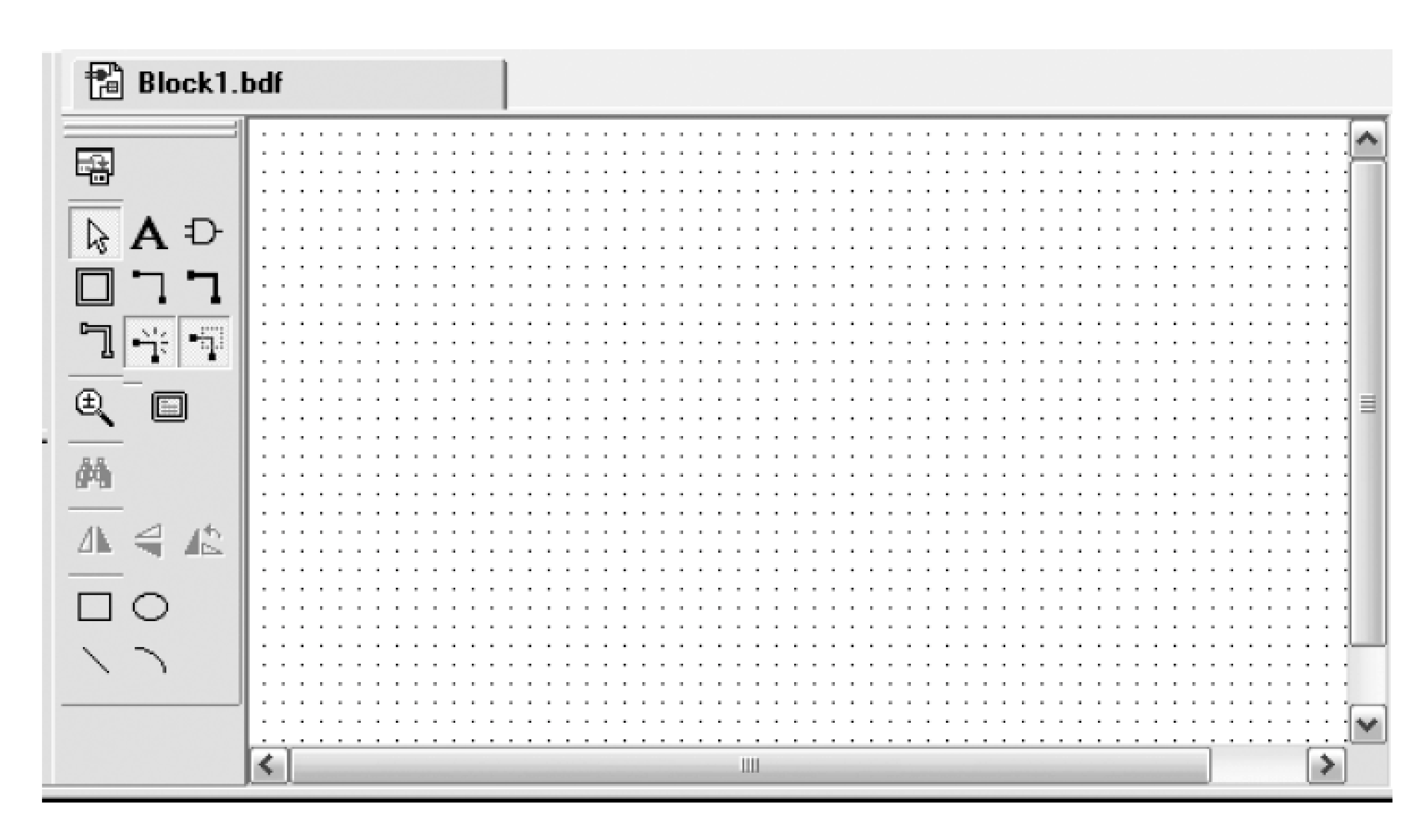


图 3-88　符号框图文件

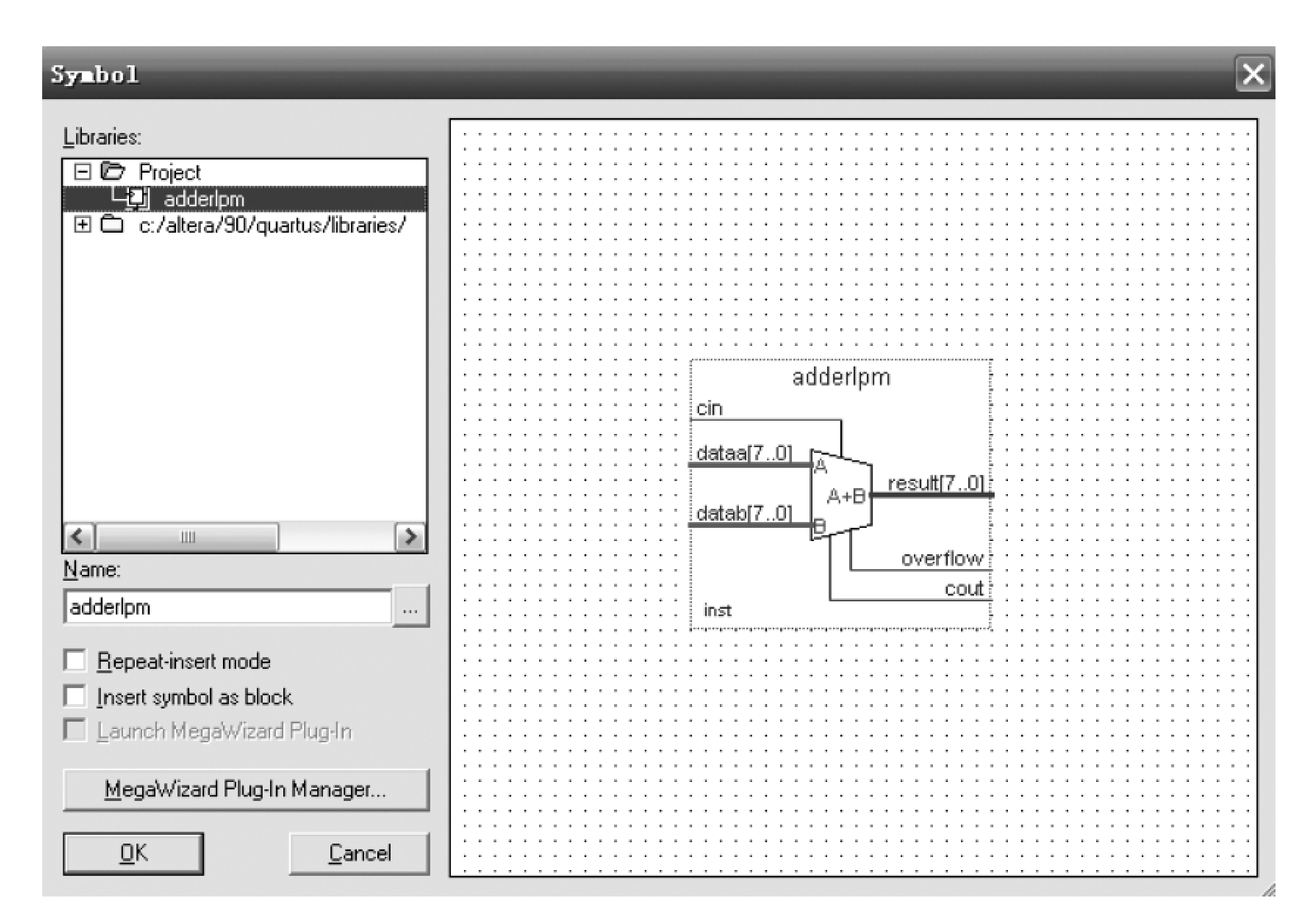


图 3-89　选择模块对话框

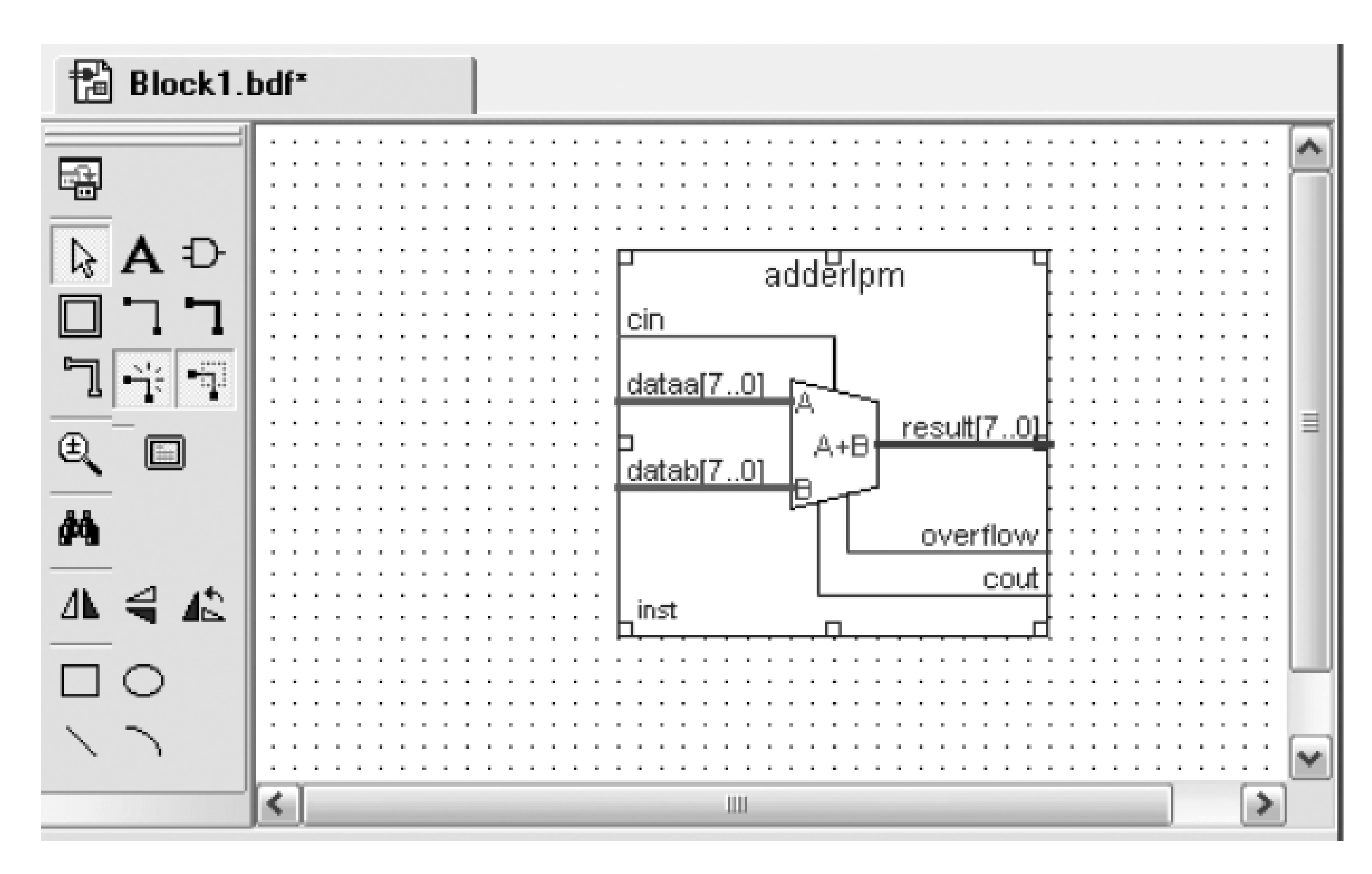


图 3-90　插入加法寄存器

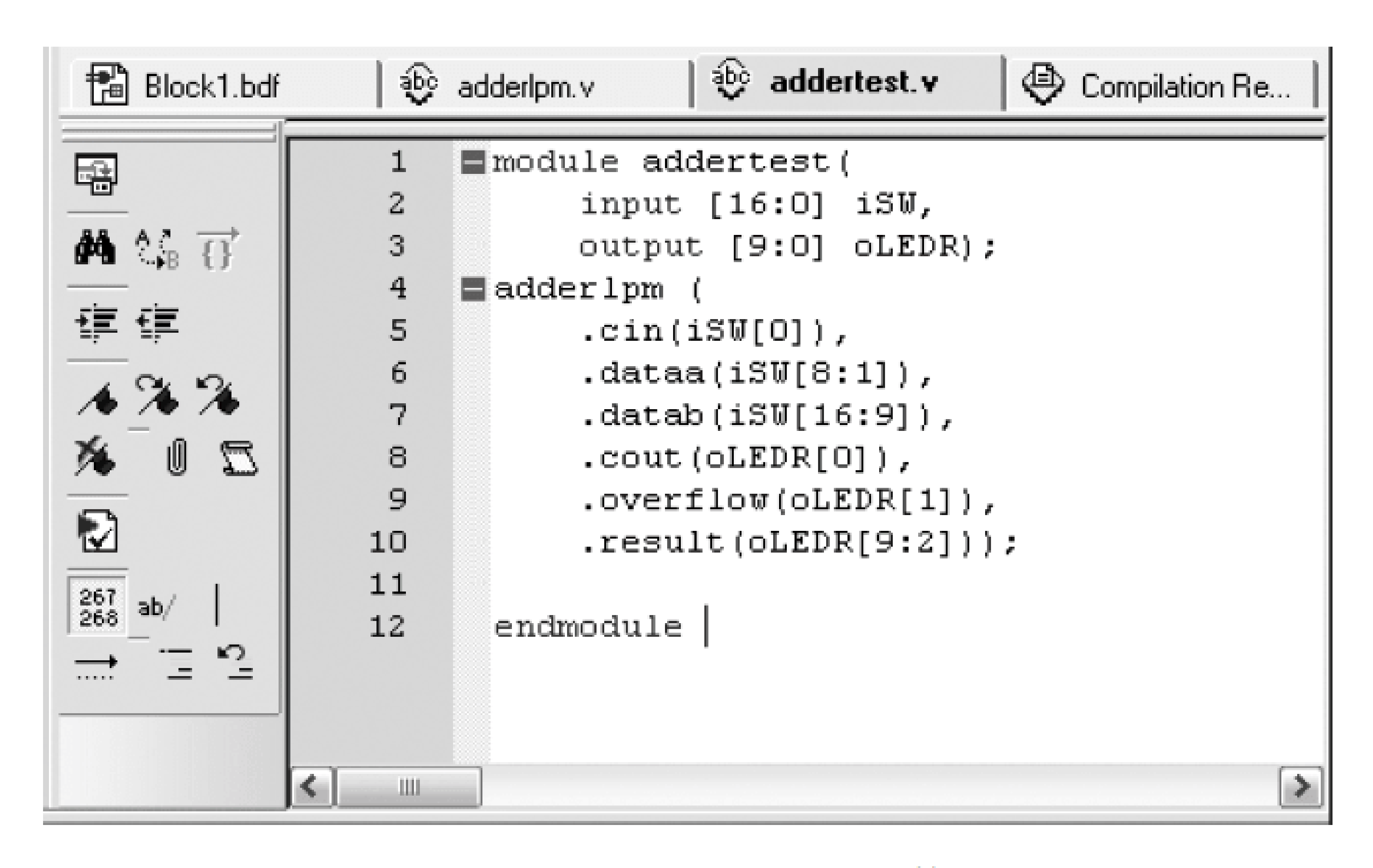


图 3-91　Verilog 顶层实体

选择合适的方式编译工程，完成编译后查看编译结果，注意本工程所占用的逻辑单元的数量，与其他方式实现加法器所用的逻辑单元数进行比较。

3.5.3　实验内容

3.5.3.1　4 位串行加法器的设计

请先用门级描述语言设计一个 1 位全加器，然后再完成 4 位串行加法器。用拨动开关作为输入端，用数码管或者发光二极管作为输出显示。将编译好的电路下载至 FPGA 芯片中，验证电路功能。

3.5.3.2　用算术赋值语句实现 8 位加法器

上述 4 位串行加法器的设计，如果推广到更多位加法器的设计，程序代码将会变得异常复杂，运行效率也将很低。在 Verilog 语言中，可以使用算术赋值语句和向量来执行这种算术运算。如果我们定义如下向量：

```
input  [n-1:0] X,Y;
output [n-1:0] S;
```

则算术赋值语句

```
S=X+Y;
```

就可以实现 n 位加法器。

注意，该代码定义了可以生成 n 位加法器的电路，但是该加法器电路并不包含进位输出信号和算术溢出信号。可以证明算术溢出信号可以用下面的表达式得到：

$$Overflow=(x_{n-1}==y_{n-1})\ \&\&\ (s_{n-1}\ !=x_{n-1})$$

请建立一个新的 Verilog 文件，其设计内容为 8 位加法器，此加法器要求带最低位的进位、有最高位进位和溢出判断位。用 Quartus Ⅱ 的 RTL Viewer 工具查看代码生成的

门级电路，然后用 Technology Map Viewer 工具查看加法器在 FPGA 中的实现，并与前个实验生成的电路进行比较。

将编译好的电路下载至 FPGA 芯片中，验证电路功能。

3.5.3.3* 简单 ALU 设计

查阅相关资料，完成一个只能进行补码加减运算的 32 位 ALU，此 ALU 的核心部件是一个 32 位加法器，能够根据控制端完成加、减和判断是否为零等运算。图 3-92 是一个参考逻辑图。输入信号是两个 32 位的数据线、一个加/减控制端。输出信号是一个 32 位的结果，以及溢出、是否为 0 和是否有进位各 1 位。

请自行设计具有此功能的 ALU，也可以参考图 3-92 设计，数据宽度可根据硬件资源调整。在可能的情况下，可以思考：如果给 ALU 添加逻辑运算功能，该如何添加？

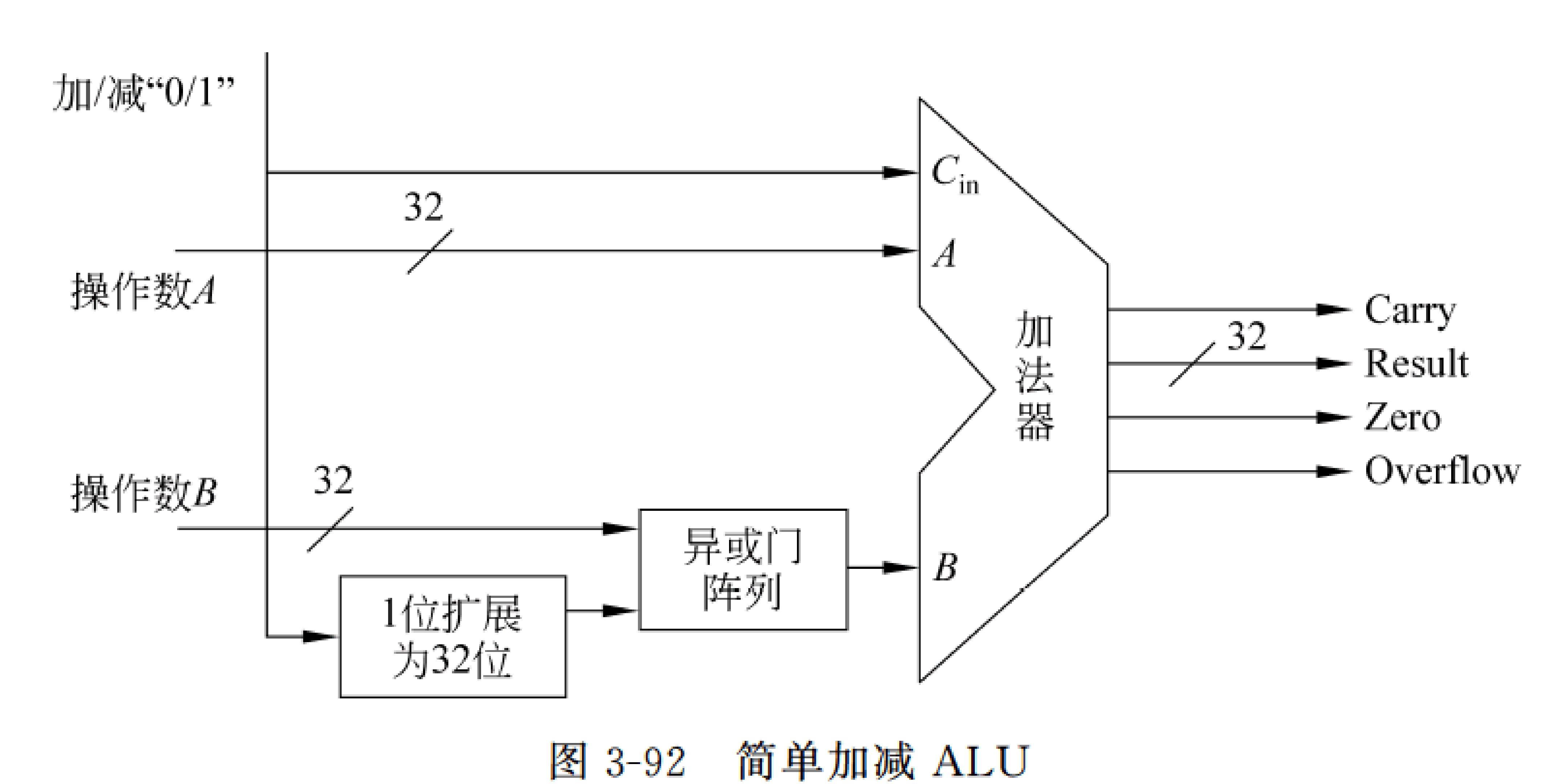


图 3-92 简单加减 ALU

3.5.3.4* 16 位乘法器设计

请查阅相关资料，选择适当算法，设计一个 16 位的乘法器，并设法验证此乘法器的功能。

第 4 章 时序逻辑电路设计

组合逻辑电路的输出只取决于当前的输入，而时序逻辑电路的输出不仅取决于当前的输入，还取决于过去的输入。

大多数时序电路的状态变化都受时钟信号控制，如果电路的状态在时钟信号为高电平或者低电平时刻发生变化，则称此电路是电平触发的；如果电路的状态在时钟信号由低电平变为高电平时刻发生变化，则称此电路是上升沿触发的；如果电路的状态在时钟信号由高电平变为低电平时刻发生变化，则称此电路是下降沿触发的。

4.1 触发器和锁存器实验

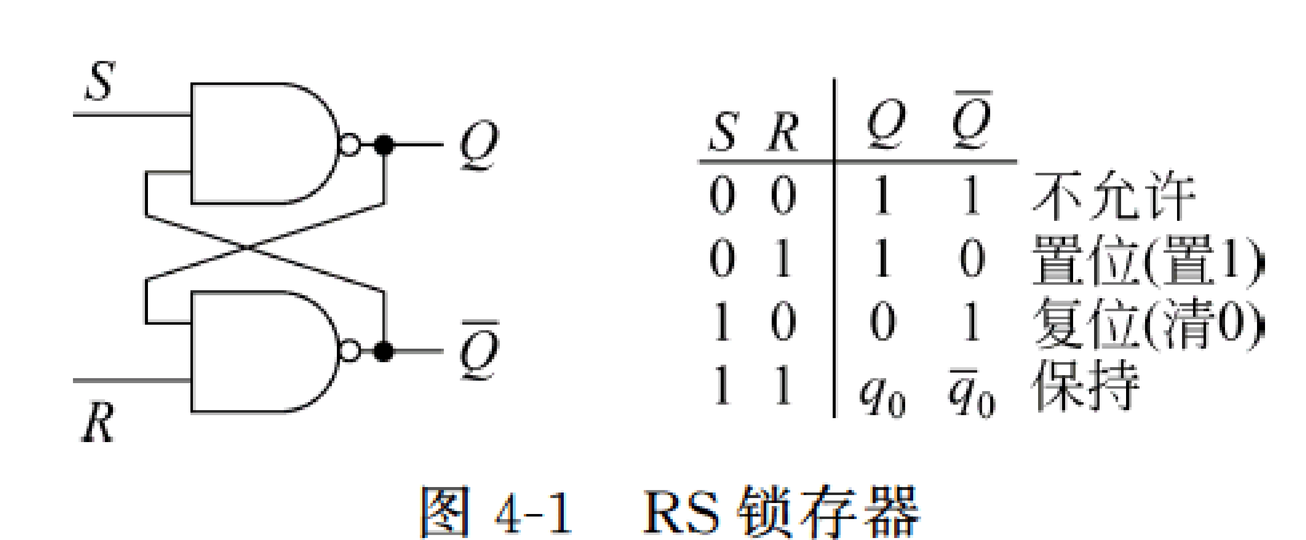
锁存器和触发器是时序电路的基本构件。锁存器和触发器都是由独立的逻辑门电路和反馈电路构成的，锁存器在时钟信号为有效电平的整个时间段，不断监测其所有的输入端，此段时间内任何满足输出改变条件的输入，都会改变输出；触发器只有在时钟信号变化的瞬间才改变输出值。

常见的有 RS 锁存器、D 锁存器、D 触发器、JK 触发器和 T 触发器等。

本次实验的目的是复习锁存器和触发器的工作原理，复习时序电路中电路时序图的分析和阅读。学习在 Quartus Ⅱ 编译器中如何对时序电路进行仿真，了解 Verilog 语言中阻塞赋值语句和非阻塞赋值语句的区别。

4.1.1 RS 锁存器

图 4-1 是一个使用与非门构成的 RS 锁存器和其真值表，根据电路原理图和真值表可以看出，RS 锁存器有 4 种不同的状态。当 R 和 S 同时为“0”时，Q 和 $\overline{Q}$ 均为“1”，这种情况是不允许的，所以是无效状态。当仅有 S 为“0”时，输出 Q 值为“1”；当仅有 R 为“0”时，输出 Q 值为“0”；当 R 和 S 全为“1”时，Q 值保持原来的值不变。由电路原理图可以分析出，只有当 R 和 S 全为“0”时，此时的锁存器值无效，其他三种状态都是稳定的，只要输入的值不改变，输出的值也保持不变。

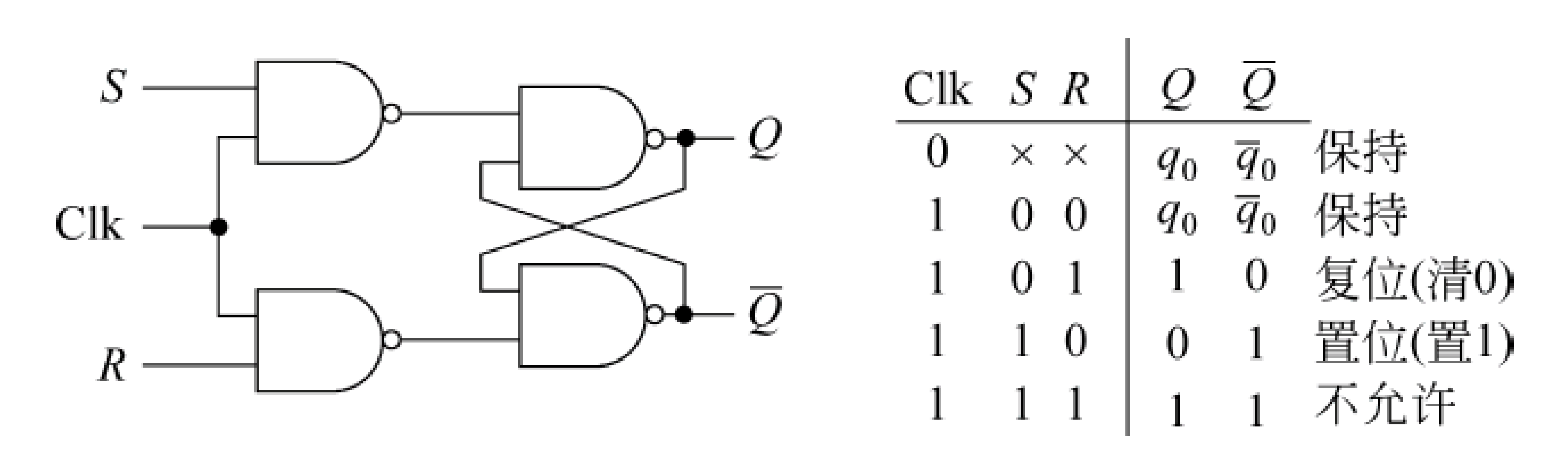


S	R	Q	$\overline{Q}$	
0	0	1	1	不允许
0	1	1	0	置位(置1)
1	0	0	1	复位(清0)
1	1	q_0	$\overline{q}_0$	保持

图 4-1 RS 锁存器

4.1.2 时钟触发的 RS 锁存器

在 RS 锁存器的基础上增加两个与非门，就可以构成由时钟信号触发的锁存器。图 4-2 中在 RS 锁存器的基础上增加了两个与非门，可以看出，当 Clk 为“0”时，这新加的两个与非门工作在“关门”状态输出恒为“1”，此时对于 RS 锁存器而言处于保持状态，当 Clk 为“1”时，新加的两个与非门工作在“开门”状态，此时 RS 锁存器的工作状态同不带时钟使触发端的 RS 锁存器相同。

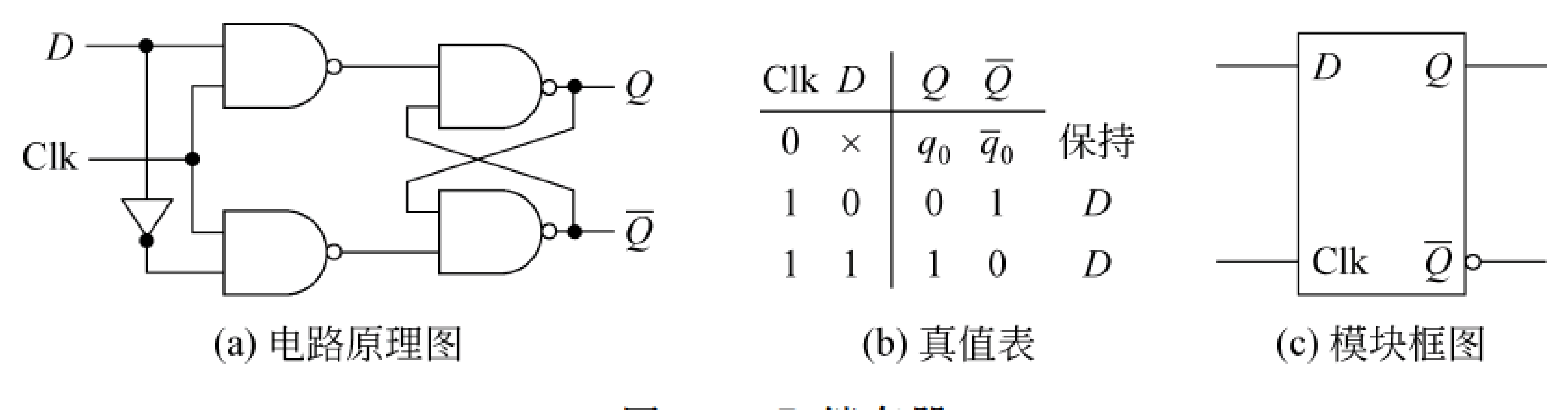


Clk	S	R	Q	$\overline{Q}$	
0	×	×	q_0	$\overline{q}_0$	保持
1	0	0	q_0	$\overline{q}_0$	保持
1	0	1	1	0	复位(清0)
1	1	0	0	1	置位(置1)
1	1	1	1	1	不允许

图 4-2 时钟触发 RS 锁存器

4.1.3 D 锁存器

RS 锁存器中有一个 Q 和 $\overline{Q}$ 同时为“1”的无效状态，这是 R 和 S 同时为“1”的缘故，如果强制 R 和 S 总是相反的逻辑，就可以避免这一现象产生。如图 4-3 所示，这个电路就是 D 锁存器电路。当时钟触发信号为“0”时，输出保持不变；当时钟触发信号为“1”时，Q 输出 D 的值，即 Q 随着 D 值的改变而改变。



Clk	D	Q	$\overline{Q}$	
0	×	q_0	$\overline{q}_0$	保持
1	0	0	1	D
1	1	1	0	D

(a) 电路原理图　(b) 真值表　(c) 模块框图

图 4-3 D 锁存器

Quartus Ⅱ编译器为设计者提供了锁存器的代码模板，编写代码时可以参考代码模板编写程序，在代码编辑区单击右键，选择“Insert Template”，在弹出的对话框中选择“Verilog HDL→Logic→Latches→Basic Latch”，单击“Insert”按钮即可。

将模板插入后，根据实际情况进行修改，编写出自己的锁存器和测试代码，如图 4-4 和图 4-5 所示。

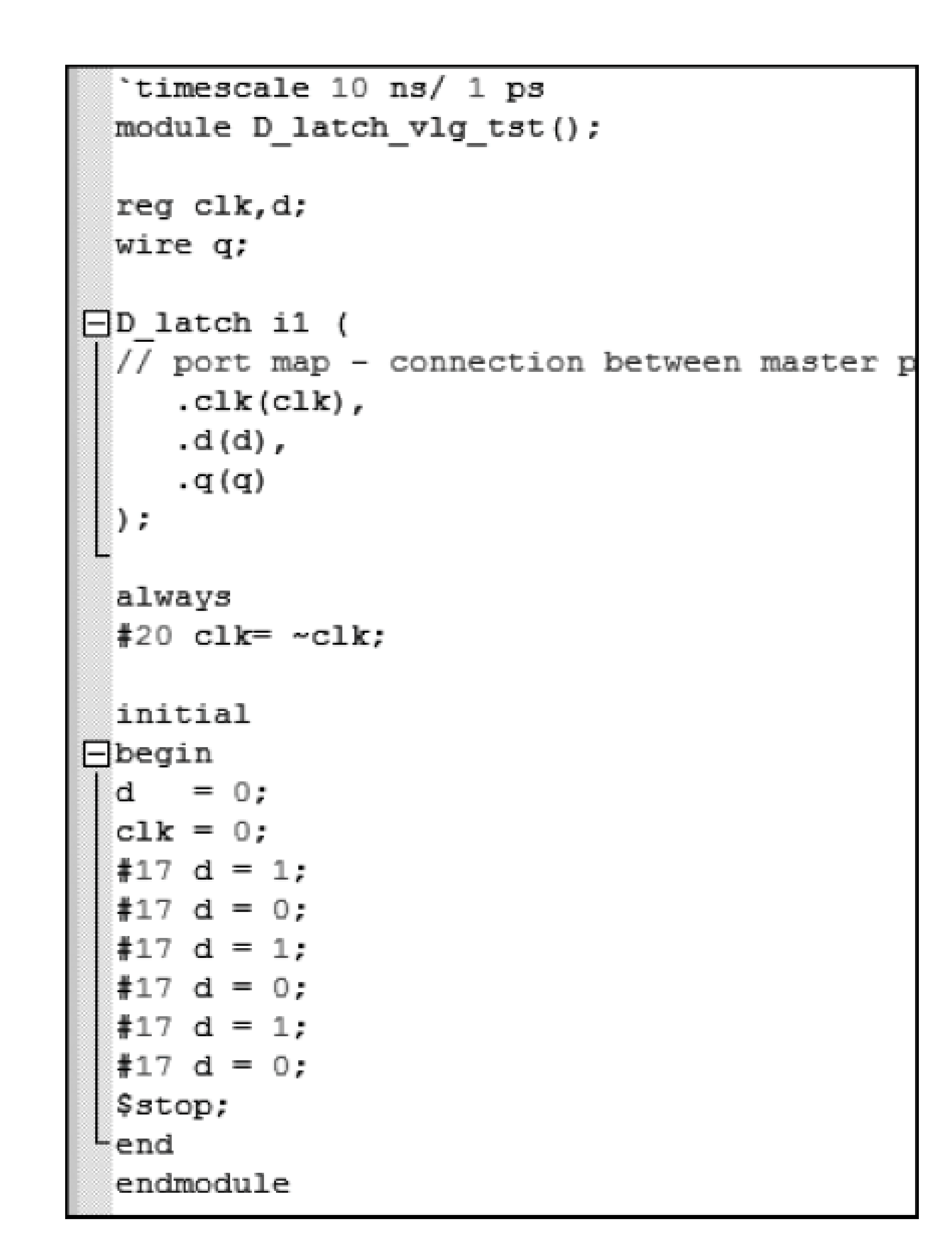
对 D 锁存器进行功能仿真，如图 4-6 所示，由图中可以看出，当 clk 信号有效(为“1”)时，输出端 q 的值随着输入端 d 的改变而改变。当 clk 无效(为“0”)时，无论 d 的值如何改变，输出端 q 的值都不改变。

前面讨论的锁存器都是在时钟为高电平(也可以是低电平)时触发的，同时我们希望锁存器只在时钟的特定时刻(如上升沿或者下降沿)触发，锁存此时刻 D 的值，这样的锁存器通常称为时钟边沿触发的触发器。

```
module D_latch(d,clk,q);
  input d,clk;
  output reg q;


always @ (*)
begin
    if (clk)  // Update the variab
    begin
        q <= d;
    end
end
endmodule
```

图 4-4　锁存器 Verilog 语言代码



```
`timescale 10 ns/ 1 ps
module D_latch_vlg_tst();

reg clk,d;
wire q;

D_latch i1 (
// port map - connection between master p
    .clk(clk),
    .d(d),
    .q(q)
);

always
#20 clk= ~clk;

initial
begin
d   = 0;
clk = 0;
#17 d = 1;
#17 d = 0;
#17 d = 1;
#17 d = 0;
#17 d = 1;
#17 d = 0;
$stop;
end
endmodule
```

图 4-5　D 锁存器测试代码

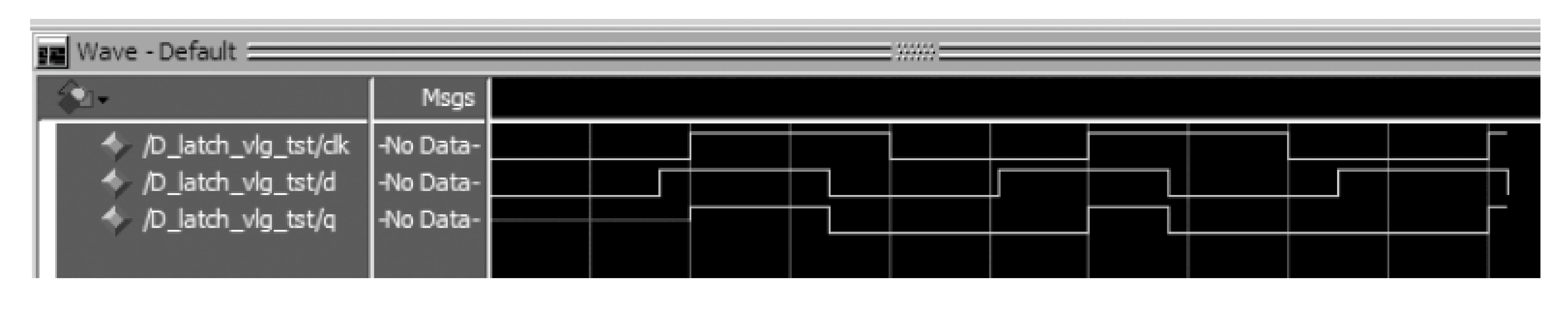
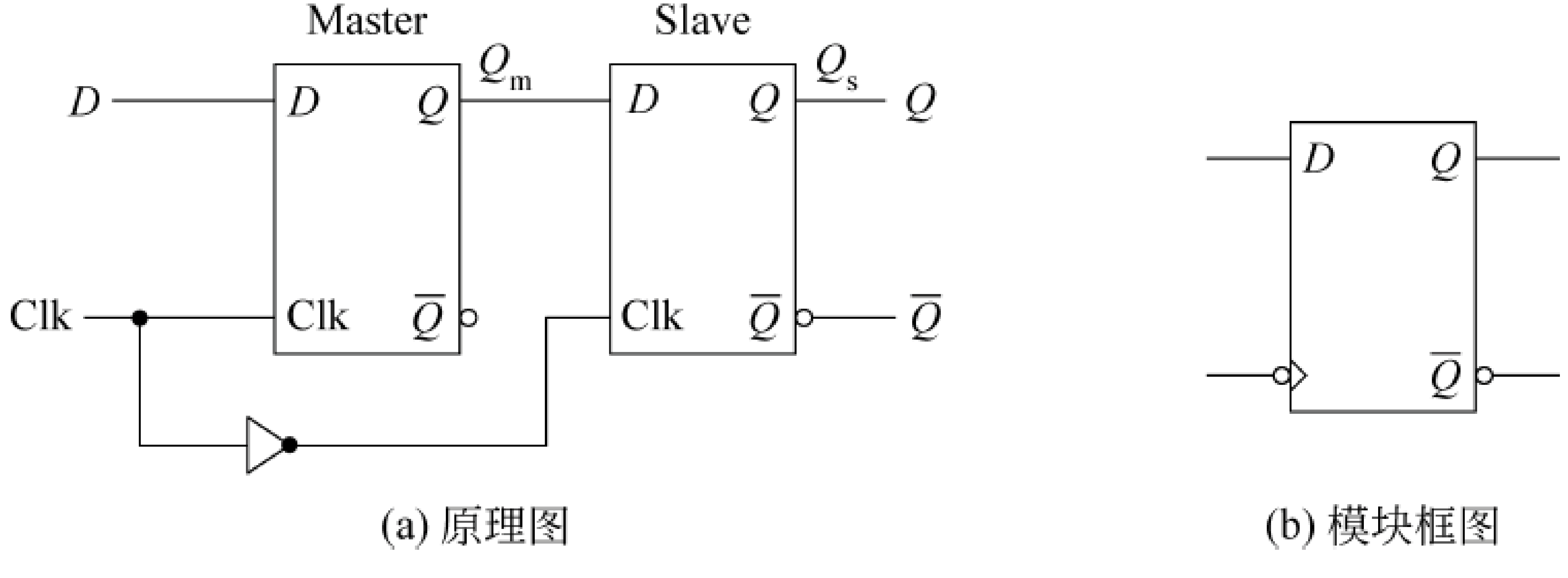


图 4-6　D 锁存器仿真图

4.1.4　时钟边沿触发的 D 触发器

用两个锁存器可以构成触发器，如图 4-7(a)所示。图中的两个 D 锁存器，前者为主(Master)锁存器，后者为从(Slave)锁存器。当 Clk 信号为“1”时，主锁存器的 Q_m 随着 D 的变化而变化，而从锁存器的状态则保持不变。当时钟信号变为“0”后，主锁存器的状态不再变化，而从锁存器的状态则跟随 Q_m 状态的变化而变化，由于当 Clk＝0 时，Q_m 不会发生变化，因此对于外部的观察者而言，在一个时钟周期内 Q 只在 Clk 从“1”变为“0”(即时钟的负跳变沿或下降沿)的时候发生一次变化。因此，我们也可以说输出信号 Q 是在



(a) 原理图　　(b) 模块框图

图 4-7　D 触发器

时钟下降沿采集到的输入信号 D 的瞬间值。D 触发器的图形符号如图 4-7(b)所示。

用 Verilog 语言实现 D 触发器非常方便,D 触发器的 HDL 代码及其测试代码如图 4-8 所示。

```
module D_trigger(d,clk,q);
  input d,clk;
  output reg q;

// Update the variable only
always @ (posedge clk)
  begin
      q <= d;
  end
endmodule
```

```
module D_trigger_vlg_tst();

reg clk,d;
wire q;

D_trigger i1 (
// port map - connection between
    .clk(clk),
    .d(d),
    .q(q)
);

always
#20 clk= ~clk;

initial
begin
d   = 0;
clk = 0;
#17 d = 1;
#17 d = 0;
#17 d = 0;
#17 d = 1;
#17 d = 1;
#17 d = 0;
$stop;
end
endmodule
```

图 4-8　D 触发器的 Verilog 代码和测试代码

对图 4-8 实现的 D 触发器进行功能仿真,如图 4-9 所示。从图中可以看出,在每个时钟信号的上升沿,q 读取此时刻 d 的值输出,一直保持到下一个时钟上升沿,下一个时钟上升沿到来时,q 再重新读取 d 的值。

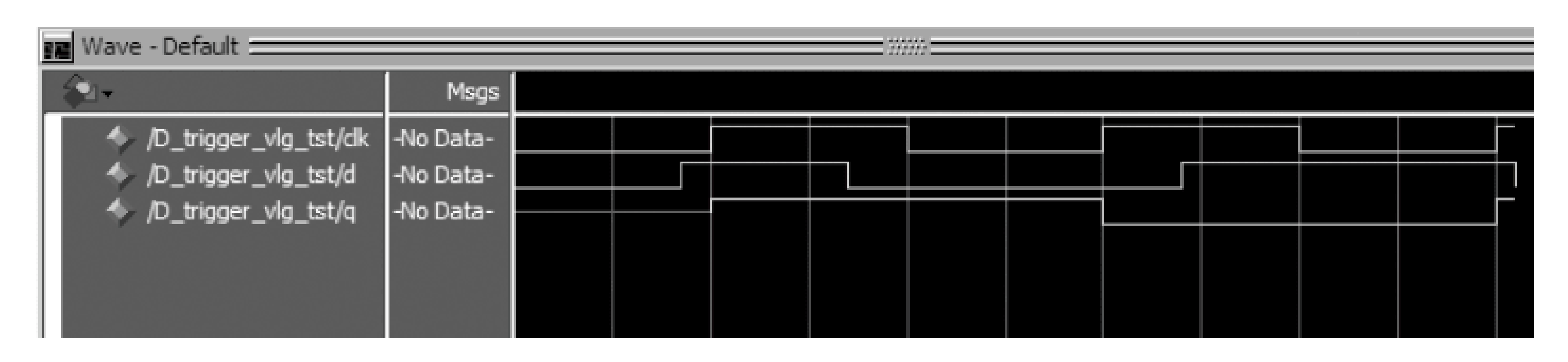


图 4-9　D 触发器功能仿真图

对电路进行仿真,用 RTL Viewer 工具查看代码生成的门级电路,然后用 Technology Map Viewer 工具查看触发器在 FPGA 中的实现。请读者自行为电路配置相应的输入和输出引脚,编辑工程,将电路下载到 DE2-70 平台上,对电路进行功能测试。

4.1.5　触发器设计中的非阻塞赋值语句

Verilog 语言中有两种赋值语句,一种赋值语句采用赋值符号"="赋值,被称为阻塞赋值语句;另一种赋值语句采用赋值符号"<="赋值,被称为非阻塞赋值语句。

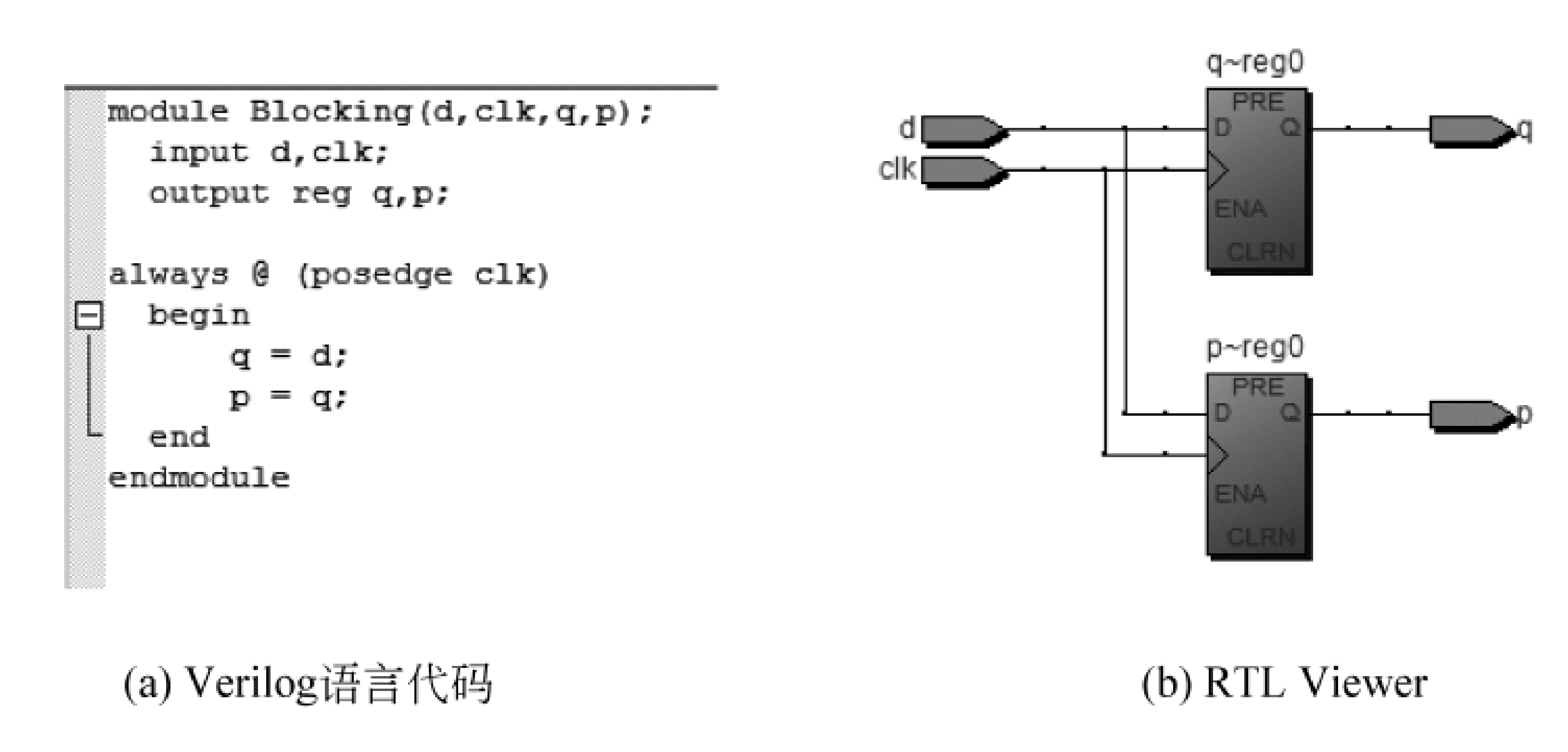
阻塞赋值语句是立即赋值语句,其形式和作用都类似于其他任何过程语言(如 C 语言)的赋值语句。阻塞赋值语句在执行时,首先计算赋值语句右边表达式的值,得到结果

后立即将值赋给左边的变量。

而非阻塞赋值语句却不同，非阻塞语句在执行时，也是立即计算赋值语句右边的表达式的值，但却不将结果立即赋值给左边的变量，而是要等到整个 always 块执行完毕后，经过一个无穷小的延时才完成赋值。

从下面的两个例子中可以看出这两种赋值语句在综合时会有何不同。

例 4.1 用阻塞赋值语句来设计两个触发器，其 Verilog 语言代码如图 4-10(a)所示。分析与综合后查看其寄存器传输级视图，如图 4-10(b)所示，从 RTL 图中可以看出，程序综合出了两个并列的触发器，p 和 q 的值同时跟着输入信号 d 值的改变而改变。



(a) Verilog语言代码　　(b) RTL Viewer

图 4-10　阻塞赋值语句设计的触发器

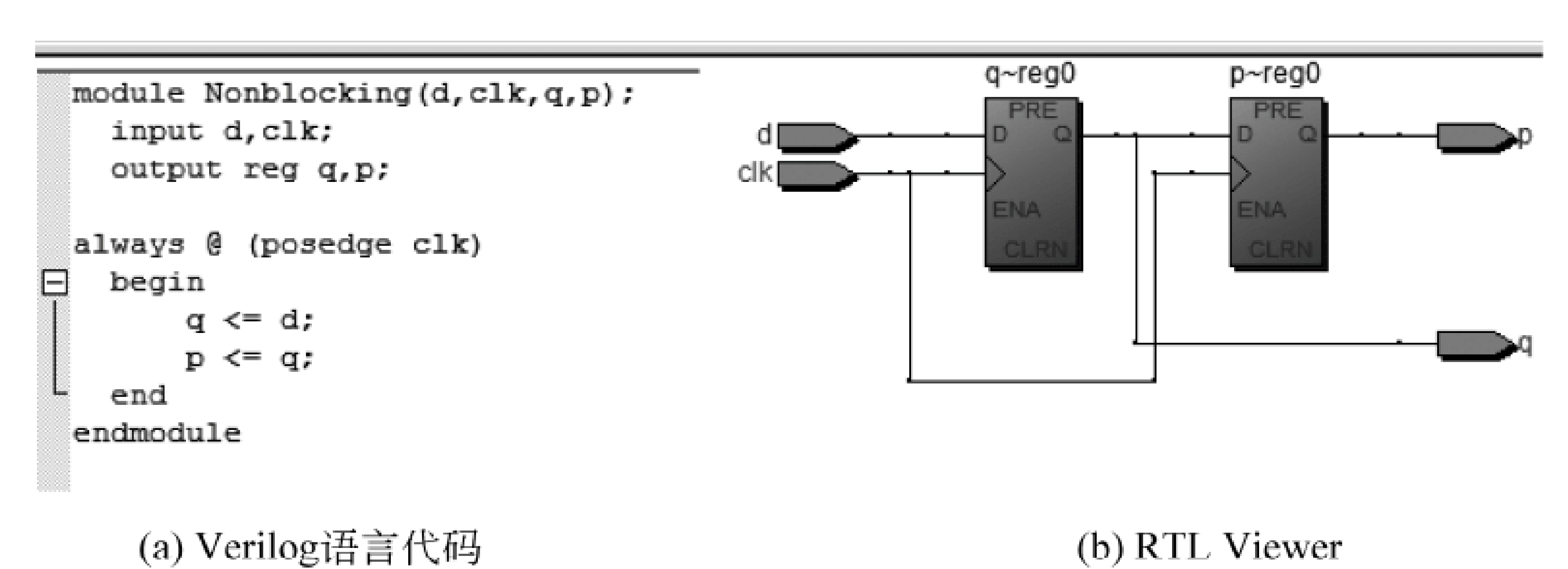
再对设计好的电路进行功能仿真，如图 4-11 所示。



图 4-11　阻塞赋值语句设计的触发器的仿真图

从图 4-11 中可以看出，在每个时钟上升沿到来时，变量 q 读取输入信号 d 的值，q 的值立即改变，同时 q 的值被送给了 p，即相当于 p 读取输入信号 d 的值。

例 4.2 用非阻塞赋值语句来设计两个触发器，其 Verilog 语言代码如图 4-12(a)所示。分析与综合后查看其寄存器传输级视图，如图 4-12(b)所示，从图中可以看出，程序



(a) Verilog语言代码　　(b) RTL Viewer

图 4-12　非阻塞赋值语句设计的触发器

综合出了两个级联的触发器。输出为 q 的触发器的输入端是 d，而输出为 p 的触发器输入信号是 q。

再对设计好的电路进行功能仿真，如图 4-13 所示。

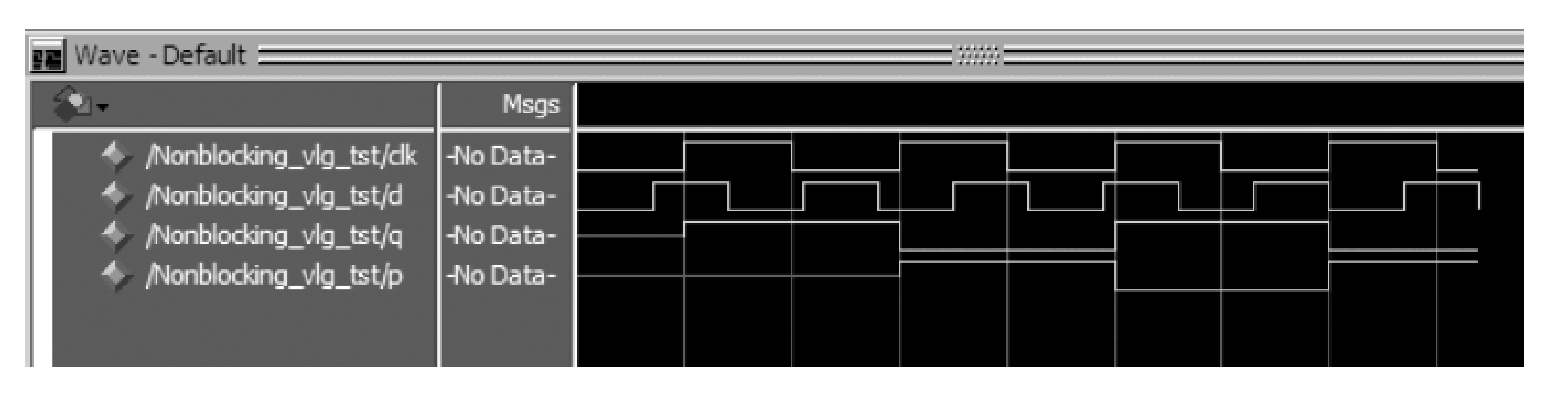


图 4-13　非阻塞赋值语句设计的触发器的仿真图

从图 4-13 中可以看出，在每个时钟上升沿到来时，进入程序 always 执行其中的语句，这时已经采样到 d 的值，但是并没有立即赋值给变量 q，所以 q 保持原来的值不变。在 always 块执行完毕后，d 的值被立即赋给 q，在 always 块中保持的 q 的原来的值被赋给了 p。到下一个时钟上升沿到来时，程序又进入 always 块中执行，always 块执行完毕时，d 的新的值被赋给了 q，原先的 q 被赋给变量 p。

4.1.6　实验内容

4.1.6.1　设计一个带有同步/异步清 0 端的 D 触发器

采用电路原理图或者 Verilog 语言代码完成设计，完成电路或者程序代码后进行分析与综合，再对此触发器进行功能仿真，分析仿真图，详述触发器的工作原理。

4.1.6.2　比较三种不同的存储单元

图 4-14 所示的电路包含了三种不同的存储单元：一个门控 D 锁存器、一个以上升沿触发的 D 触发器和一个以下降沿触发的 D 触发器。

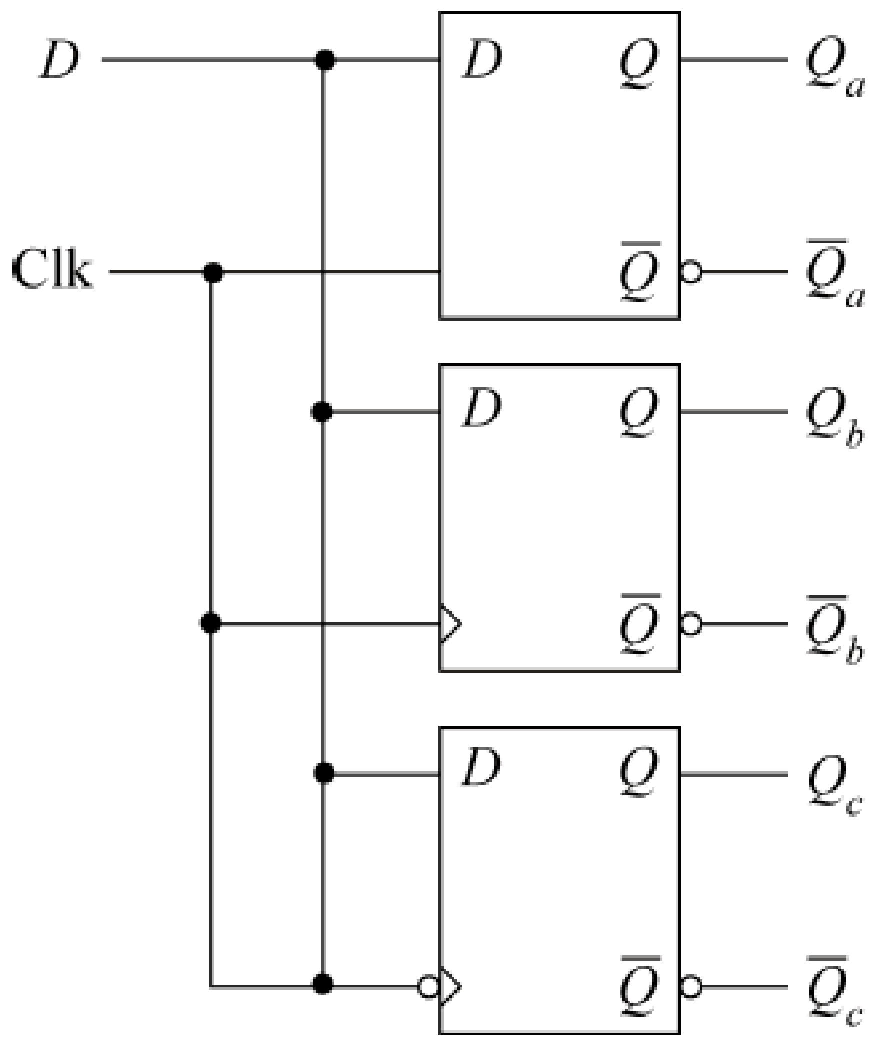


图 4-14　三种不同的存储单元

新建一个 Verilog 语言文件，在此 Verilog 语言文件中试调用三种不同的存储单元来完成。编译工程，用 RTL Viewer 工具查看代码生成的寄存器传输级电路，然后用 Technology Map Viewer 工具查看电路在 FPGA 中的实现，比较三个电路，看看有什么区别。

使用 Modelsim 对电路进行仿真，查看三个存储单元的输出结果，比较三种存储单元的不同。

4.2　寄存器实验

寄存器也是最常用的时序逻辑电路，是一种存储电路。寄存器的存储电路是由锁存器或触发器构成的，因为一个锁存器或触发器能存储 1 位二进制数，所以由 N 个锁存器

或触发器可以构成 N 位寄存器。

本实验通过介绍几种常用寄存器的设计方法，复习寄存器的原理，学习寄存器和常用的移位寄存器的设计。

4.2.1 寄存器

D 触发器可以用于存储比特信号，给 D 触发器加上置数功能就变成了 1 位寄存器，如图 4-15 所示。由图中可以看出，如果 load 信号为“1”，则输入信号 in 被送入或门中，或门的另一个输入端为“0”，此时 $D=\text{in}$，所以在下一个时钟里 $q=\text{in}$。当 load 值为“0”时，q 值被反馈到或门中，或门的另一个输入值为“0”，此时 $D=q$，因此在下一个时钟周期里 q 值保持先前的值不变。

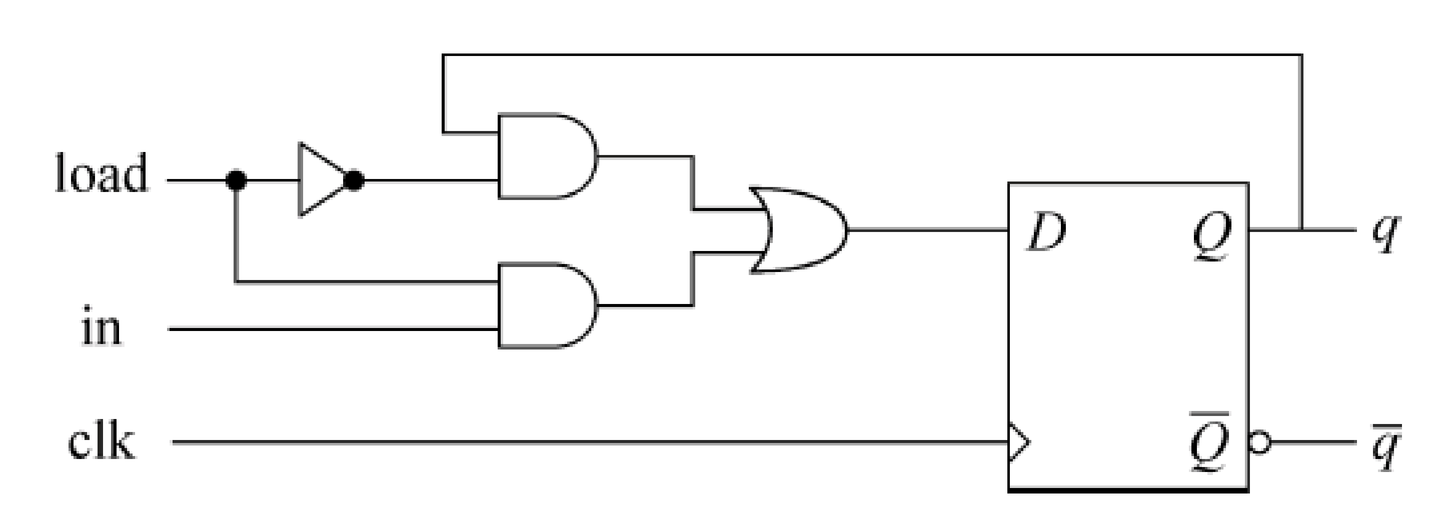


图 4-15　1 位寄存器

用 Verilog 语言设计寄存器也很简单，如程序清单 4.1 所示。

程序清单 4.1　1 位寄存器。

```
module register1(load,clk,clr,inp,q);
input  load,clr,clk,inp;
output reg q;
always @ (posedge clk)
if (clr==1)
   q<=0;
else if (load==1)
          q<=inp;

endmodule
```

由程序清单 4.1 的程序的仿真图如图 4-16 所示。

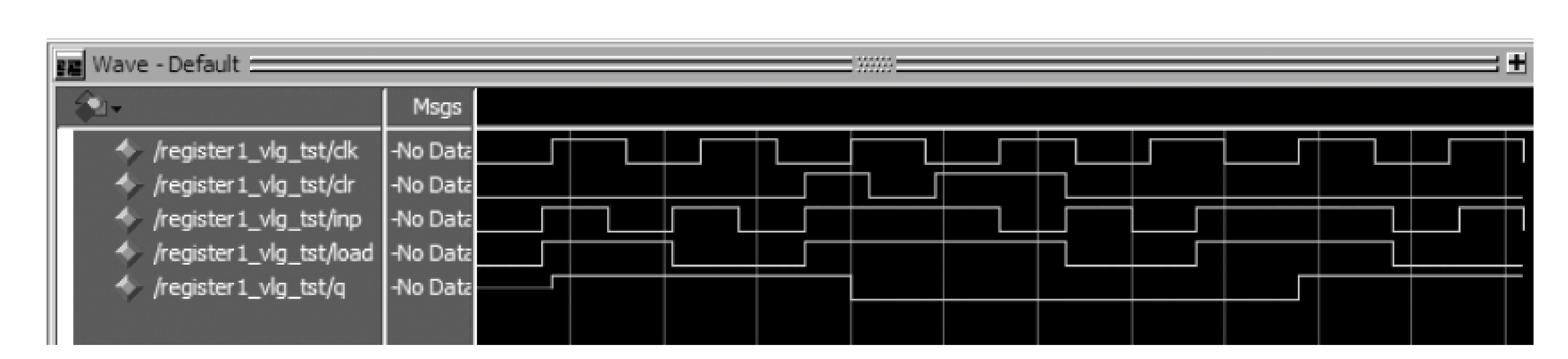


图 4-16　1 位寄存器仿真图

本例实现的是一个带有清“0”端的 1 位寄存器，还有的寄存器带有置位（置“1”）端，

图 4-17 是同时带有清“0”端和置“1”端的寄存器的逻辑示意图,读者可自行设计此寄存器。

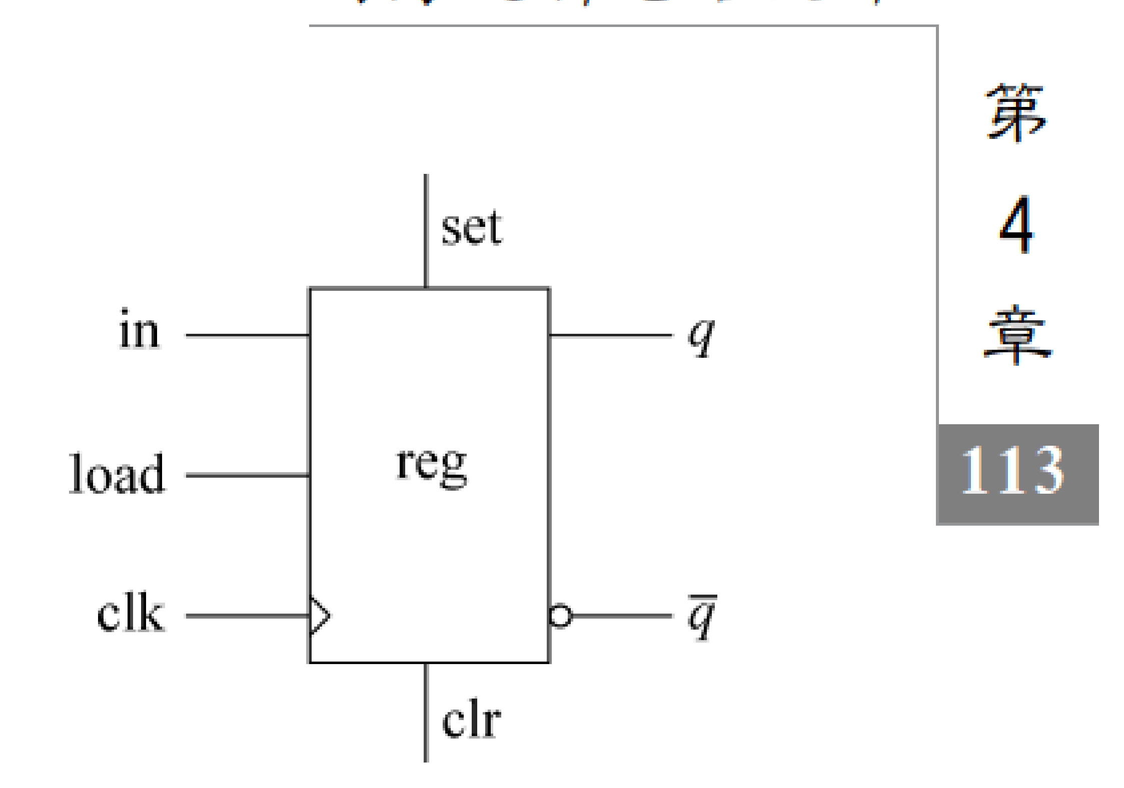


图 4-17　1 位寄存器逻辑图

将两个或者两个以上的 1 位寄存器组合在一起,这些寄存器共用一个时钟信号,就构成了多位寄存器,寄存器常被用在计算机中存储数据,例如指令寄存器、数据寄存器等。程序清单 4.2 就是利用 Verilog 语言设计多位寄存器的例子。

程序清单 4.2　4 位寄存器。

```
module register4(load,clk,clr,d,q);
input  load,clr,clk;
input  [3:0] d;
output reg [3:0] q;
always @ (posedge clk)
if (clr==1)
   q<=0;
else if (load==1)
          q<=d;
endmodule
```

程序清单 4.2 实现的电路的功能仿真图如图 4-18 所示。

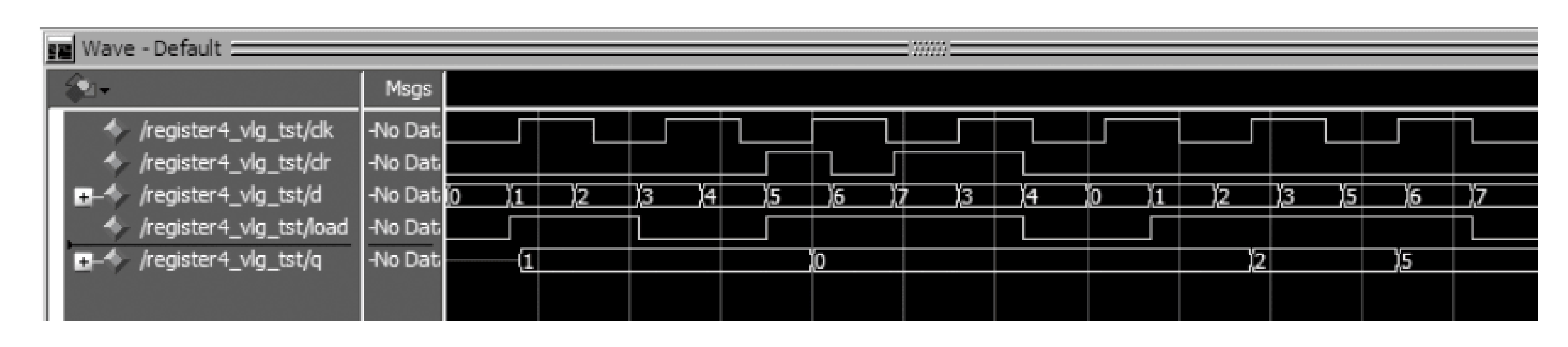


图 4-18　4 位寄存器仿真图

4.2.2　移位寄存器

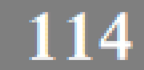
移位寄存器也是一类寄存器,它在时钟的触发沿,根据其控制信号,将存储在其中的数据向某个方向移动一位。移位寄存器也是数字系统的常用器件。

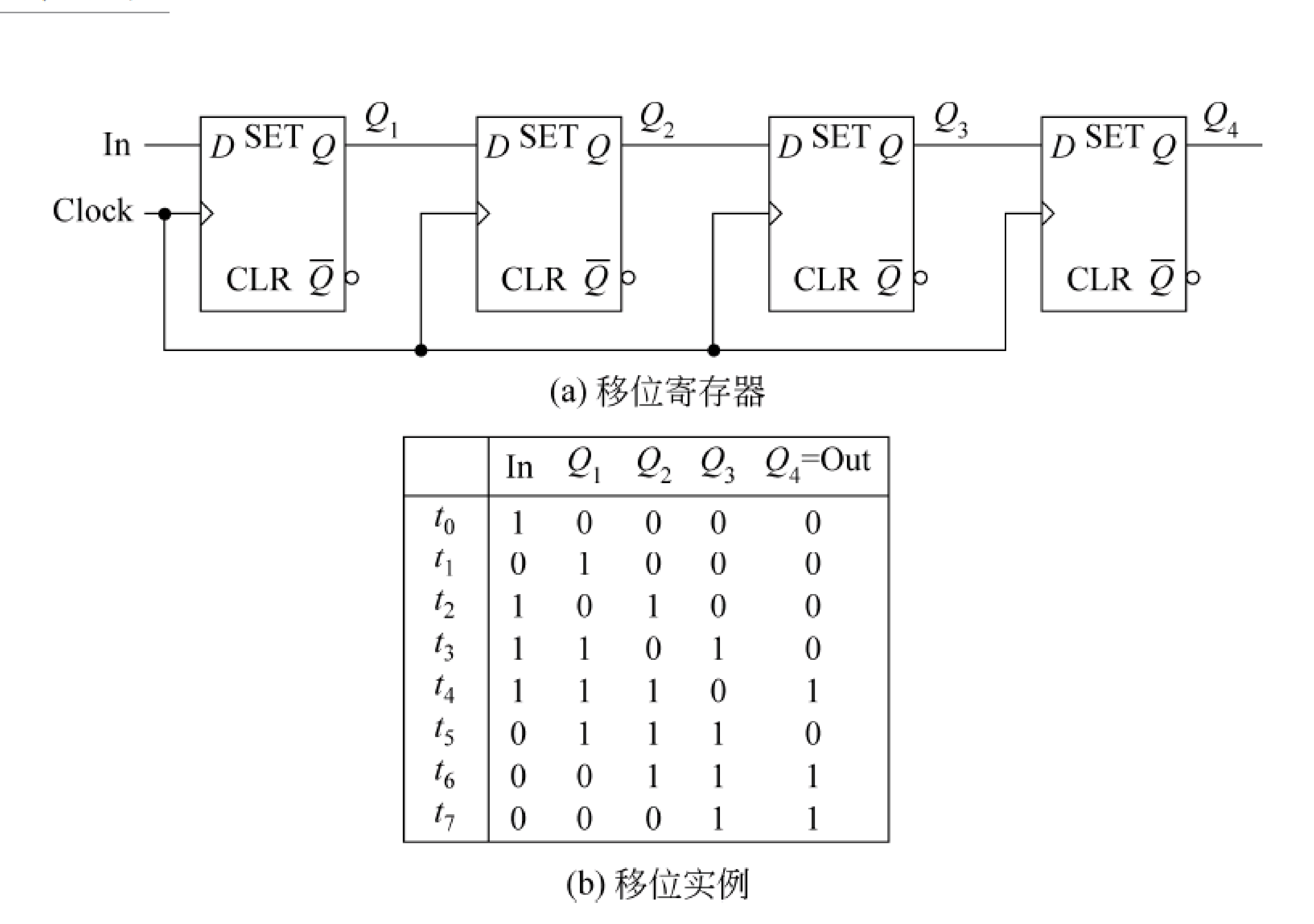
图 4-19(a)中是一个由 4 个 D 触发器构成的简单向右移位寄存器,数据从移位寄存器的左端输入,每个触发器的内容在时钟的正跳变沿(上升沿)将数据传到下一个触发器。图 4-19(b)是一个此移位寄存器的序列传递实例。

4.2.3　实验内容

4.2.3.1　简单移位寄存器

一个触发器可以存储 1 位信息,n 个触发器组合在一起就可以存储 n 位信息,称为一个寄存器。





(a) 移位寄存器

	In	Q_1	Q_2	Q_3	Q_4=Out
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	1	1	1	0	1
t_5	0	1	1	1	0
t_6	0	0	1	1	1
t_7	0	0	0	1	1

(b) 移位实例

图 4-19　移位寄存器

请设计一个 1 位 D 触发器，利用此 D 触发器完成一个 8 位的左移移位寄存器，编译工程，用 RTL Viewer 工具查看代码生成的门级电路，然后用 Technology Map Viewer 工具查看移位寄存器在 FPGA 中的实现。对电路进行仿真并下载到 DE2-70 开发板上验证其功能。

4.2.3.2　并行存取的移位寄存器

在数字系统中经常用到很多其他的移位寄存器，上例实验完成的是一个串行输入、串行输出的移位寄存器，请根据以前实验所学知识，设计一个移位寄存器，要求完成以下功能的一项或几项：

- 串行输入并行输出；
- 并行输入串行输出；
- 并行输入并行输出；
- 带清零功能。

4.2.3.3　算术移位和逻辑移位寄存器

逻辑移位不管是向左移位还是向右移位都是空缺处补 0。这里的算术移位是指考虑到符号位的移位，算术移位要保证符号位不改变，算术左移与逻辑左移相同，算术右移位左面的空位补符号位。用 Verilog HDL 语言很容易描述出移位寄存器，例如：

```
Q<= {Q[0],Q[7: 1]};                    //循环右移
Q<= {Q[7],Q[7: 1]};                    //算术右移
```

请根据表 4-1，用 Verilog HDL 语言设计一个移位寄存器，并进行仿真，查看移位寄存器的功能，将编译好的文件下载到 FPGA 开发平台上进行验证。

表 4-1 移位寄存器工作方式

控制位			工作方式
0	0	0	清 0
0	0	1	置数
0	1	0	逻辑右移
0	1	1	逻辑左移
1	0	0	算术右移
1	0	1	左端串行输入,并行输出
1	1	0	循环右移
1	1	1	循环左移

4.3 计数器实验

在数字系统中,常用计数器来记录系统的工作状态,本实验复习了计数器的工作原理,通过介绍几种简单计数器的工作过程和设计方法,学习计数器的设计。

4.3.1 加法计数器

利用触发器可以构成简单的计数器。图 4-20 是由三个上升沿触发的 D 触发器组成的 3 位二进制异步加法计数器,即在每个 Clock 的上升沿,计数器输出 $Q_2Q_1Q_0$ 加 1,“clr”是“清零”端。也可以给 D 触发器加上“置数”端,构成一个可以清零和置数的二进制异步加法计数器。

图 4-21 是此 3 位二进制异步加法计数器的状态转移图。

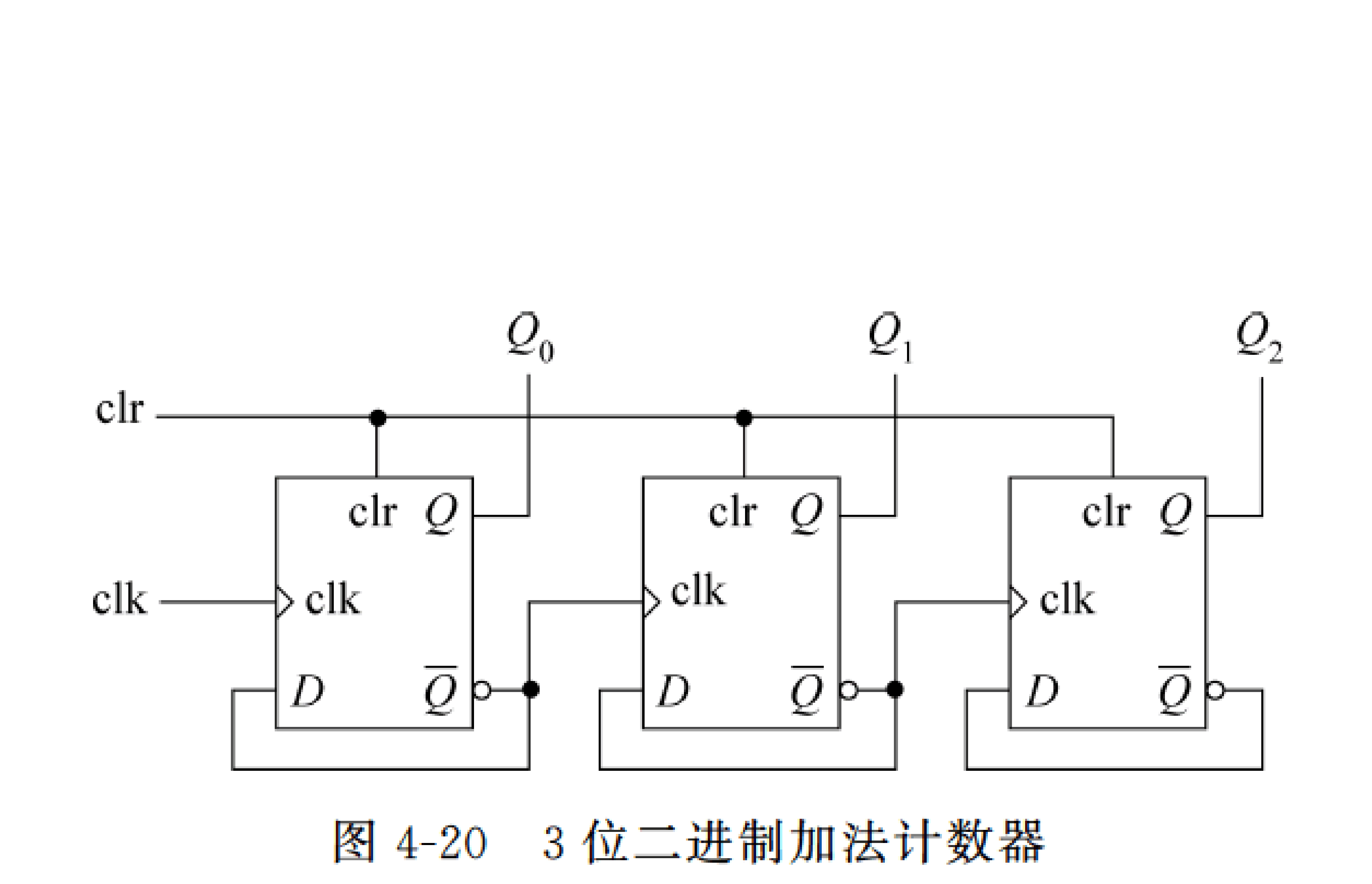


图 4-20 3 位二进制加法计数器

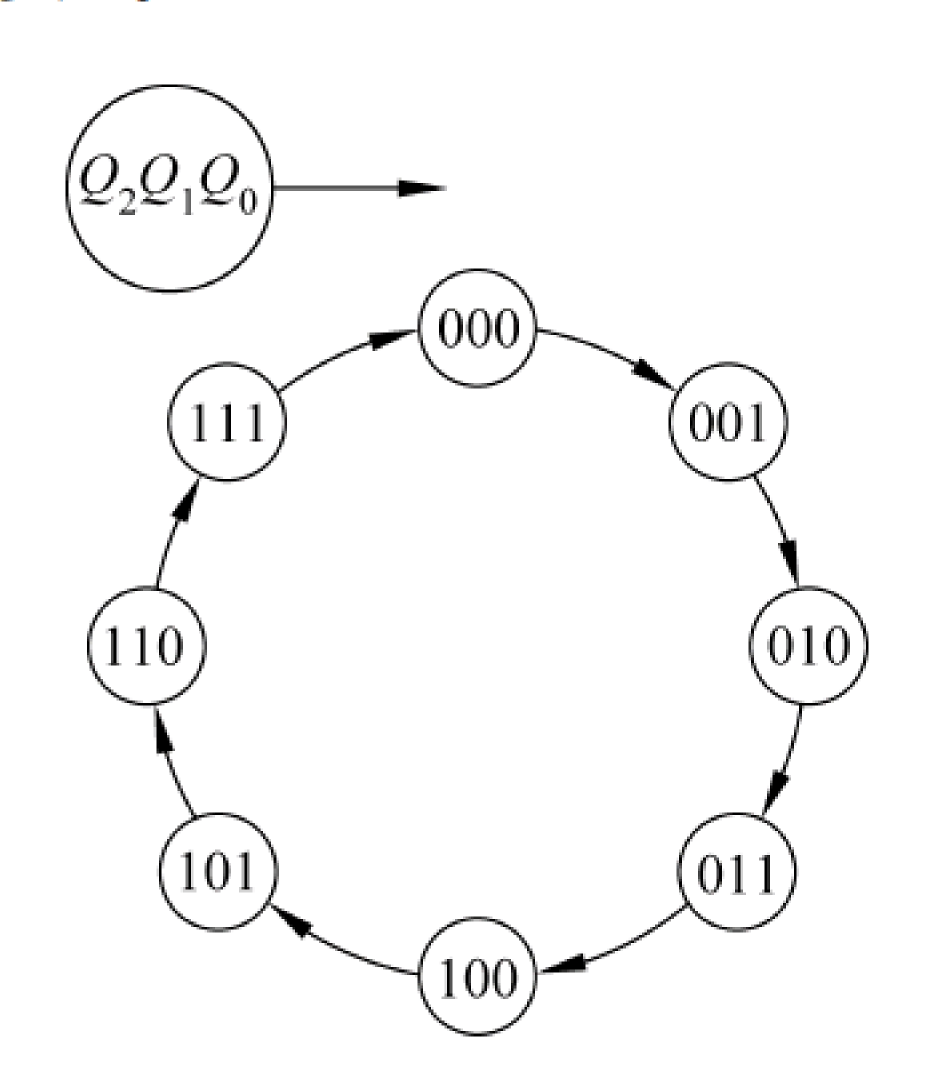


图 4-21 3 位二进制加法计数器状态图

这个 3 位二进制异步加法计数器的功能仿真图如图 4-22 所示。

4.3.2 减法计数器

利用 D 触发器同样可以构成减法计数器,图 4-23 是由三个上升沿触发的 D 触发器组成的 3 位二进制异步减法计数器。

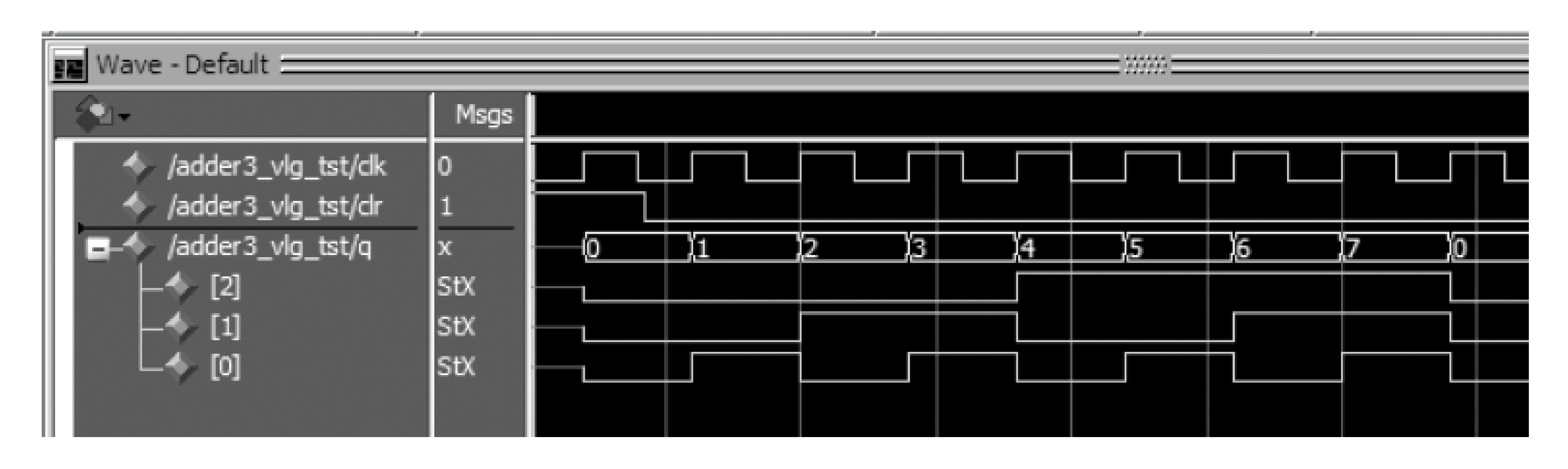


图 4-22　二进制加法计数器仿真图

图 4-24 是此 3 位二进制异步减法计数器的状态图。

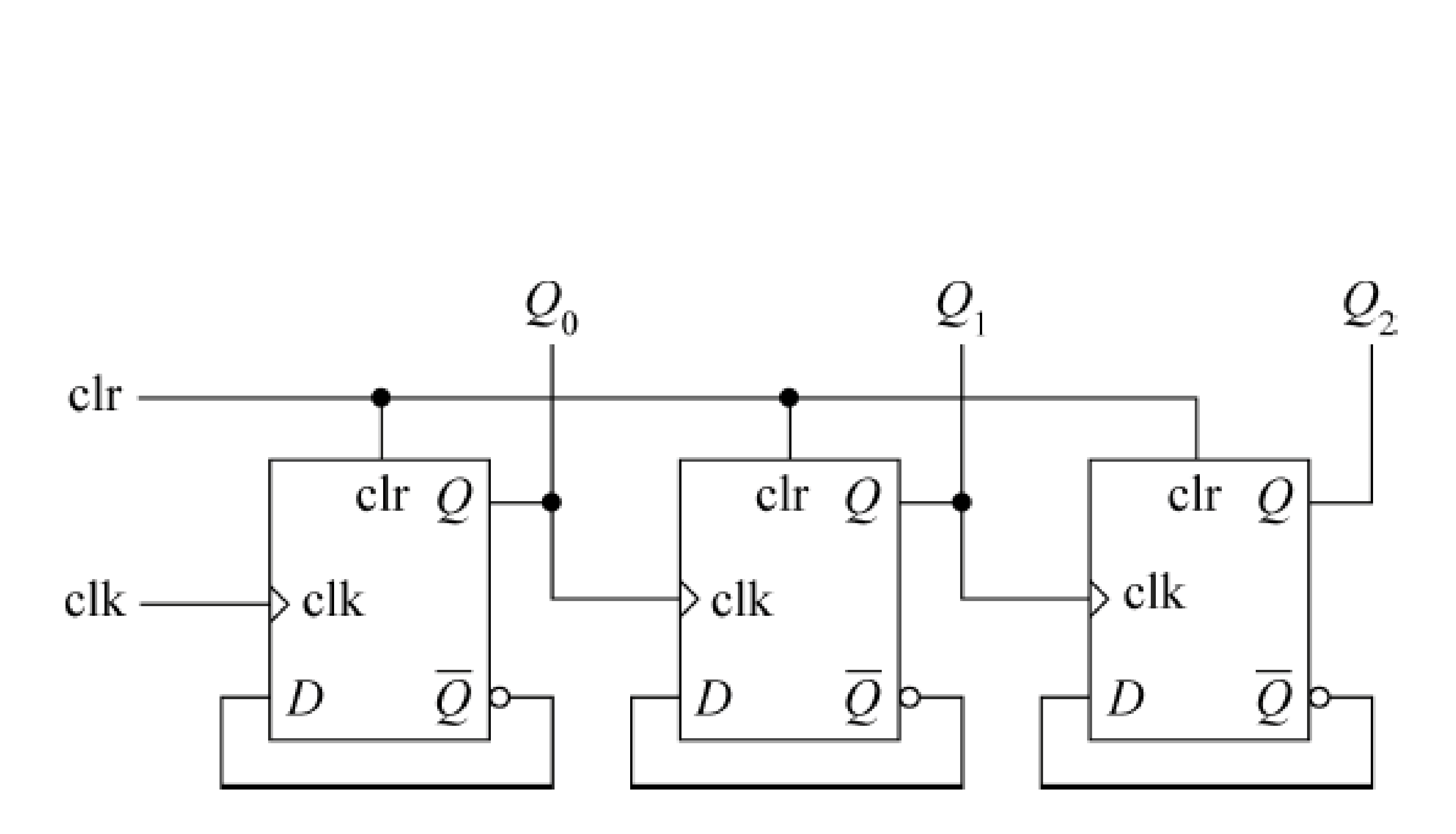


图 4-23　3 位二进制异步减法计数器

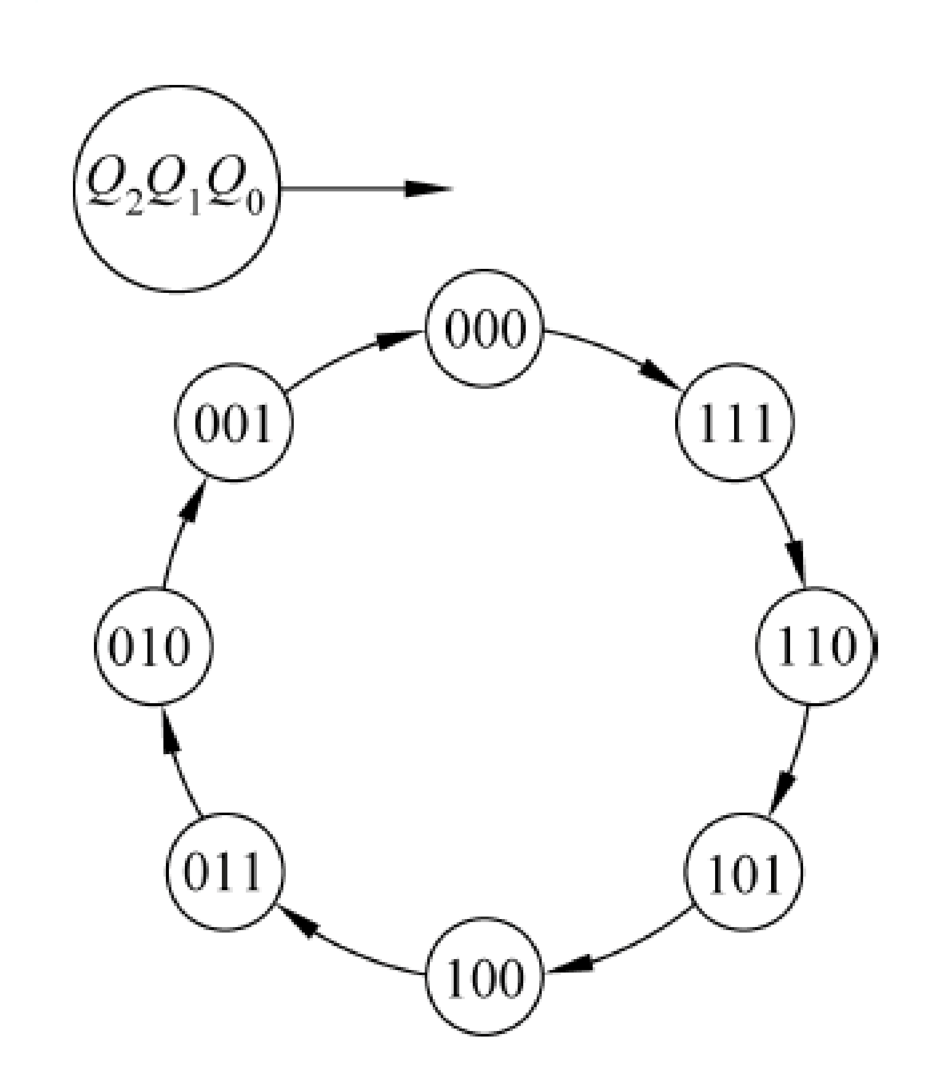


图 4-24　3 位二进制异步减法计数器状态图

利用 Verilog 语言也可以方便地构建计数器，程序清单 4.3 是一个 3 位二进制减法计数器，也可以用类似的代码构成加法计数器。

程序清单 4.3　3 位二进制减法计数器。

```
module vminus3(clk,clr,q);
    input  clk,clr;
    output reg [2:0] q;
  always @ (posedge clk)
  if(clr)
      begin q= 0;end
  else
      begin q<=q-1;end
endmodule
```

程序清单 4.3 构建的减法计数器的仿真图如图 4-25 所示。

4.3.3　实验内容

4.3.3.1　用 Verilog HDL 赋值语句实现 74X163 计数器

用 Verilog HDL 定义计数器非常简单，只要用 $Q<=Q+1$ 就可以简单地实现一个

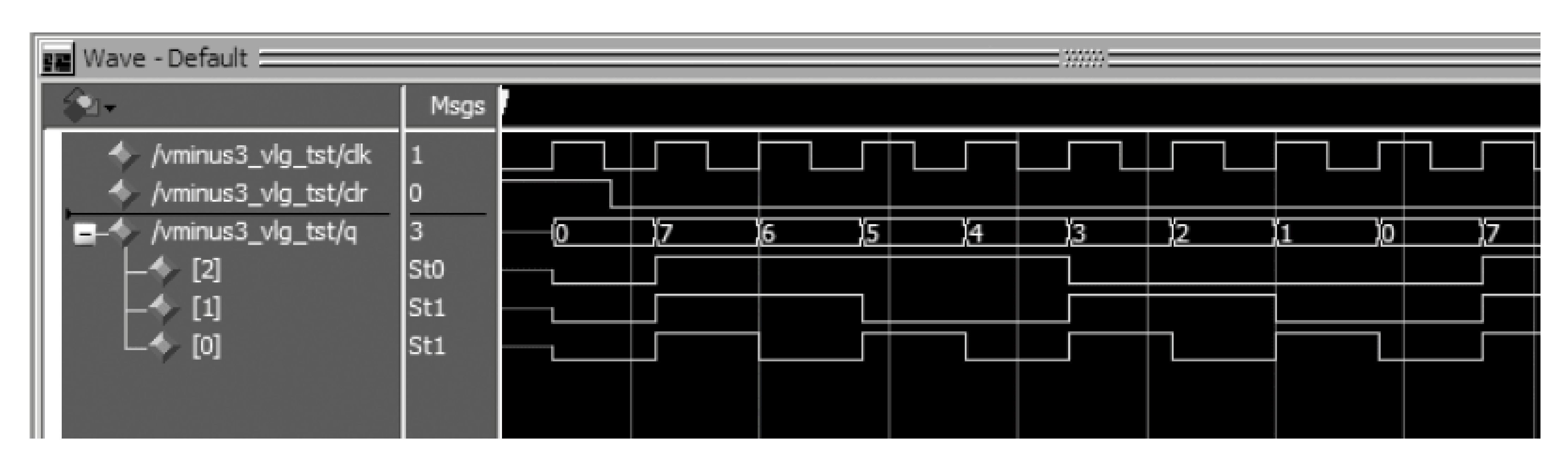


图 4-25　减法计数器仿真图

4 位二进制加一计数器。

74X163 是常用的计数器集成芯片，其功能表和逻辑符号如表 4-2 和图 4-26 所示。

请用 Verilog HDL 实现这个计数器，编译工程，用 RTL Viewer 工具查看代码生成的门级电路，然后用 Technology Map Viewer 工具查看计数器在 FPGA 中的实现。对电路进行仿真并下载到 DE2-70 开发板上，验证其功能。

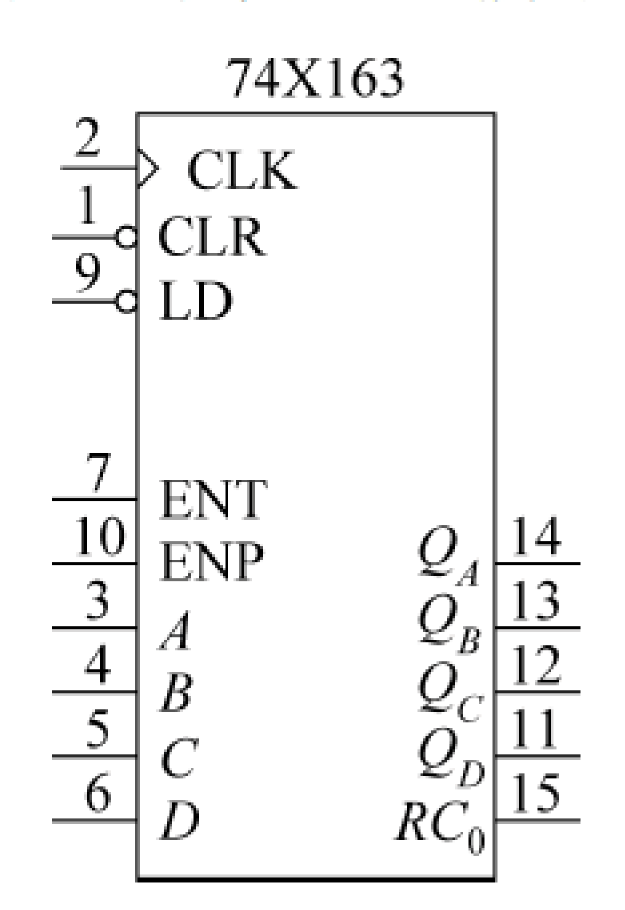


图 4-26　74X163 的逻辑符号

4.3.3.2　用参数化功能模块实现计数器

用参数化功能模块实现(LPM)一个 4 位的计数器，配置要求与上一个用 $Q<=Q+1$ 赋值语句实现的计数器一样。和上面的设计方法产生的结果进行比较，查看有何不同。

表 4-2　4 位二进制计数器 74X163 状态表

输入									当前状态				输出				
CLR_L	LD_L	ENT	ENP	CLK	D	C	B	A	Q_D	Q_C	Q_B	Q_A	Q_D	Q_C	Q_B	Q_A	RC_0
0	×	×	×	↑	×	×	×	×	×	×	×	×	0	0	0	0	0
1	0	×	×	↑	×	×	×	×	×	×	×	×	D	C	B	A	0
1	1	0	×	↑	×	×	×	×	×	×	×	×	Q_D	Q_C	Q_B	Q_A	0
1	1	×	0	↑	×	×	×	×	×	×	×	×	Q_D	Q_C	Q_B	Q_A	0/1
1	1	1	1	↑	×	×	×	×	0	0	0	0	0	0	0	1	0
1	1	1	1	↑	×	×	×	×	0	0	0	1	0	0	1	0	0
1	1	1	1	↑	×	×	×	×	0	0	1	0	0	0	1	1	0
1	1	1	1	↑	×	×	×	×	0	0	1	1	0	1	0	0	0
1	1	1	1	↑	×	×	×	×	0	1	0	0	0	1	0	1	0
1	1	1	1	↑	×	×	×	×	0	1	0	1	0	1	1	0	0
1	1	1	1	↑	×	×	×	×	0	1	1	0	0	1	1	1	0
1	1	1	1	↑	×	×	×	×	0	1	1	1	1	0	0	0	0
1	1	1	1	↑	×	×	×	×	1	0	0	0	1	0	0	1	0
1	1	1	1	↑	×	×	×	×	1	0	0	1	1	0	1	0	0
1	1	1	1	↑	×	×	×	×	1	0	1	0	1	0	1	1	0
1	1	1	1	↑	×	×	×	×	1	0	1	1	1	1	0	0	0
1	1	1	1	↑	×	×	×	×	1	1	0	0	1	1	0	1	0
1	1	1	1	↑	×	×	×	×	1	1	0	1	1	1	1	0	0
1	1	1	1	↑	×	×	×	×	1	1	1	0	1	1	1	1	1
1	1	1	1	↑	×	×	×	×	1	1	1	1	0	0	0	0	0

4.4 定　时　器

如果在计数器的时钟输入端输入一个固定频率的时钟，那么计数器就变成了计时器。

本实验的目的是学习 FPGA 开发平台上时钟源的使用，并结合计数器的设计方法学习计时器的设计。

4.4.1 开发板上的时钟信号

DE2-70 开发板有两个分别产生 28.86MHz 和 50MHz 时钟信号的振荡器。两个时钟信号都连接到 FPGA，可以为用户的逻辑电路提供时钟信号。

FPGA 上与时钟相连的 I/O 引脚在表 4-3 中列出。

表 4-3　时钟输入引脚分配

信　号　名	FPGA 引脚编号	描　　述
iCLK_28	PIN_E16	28MHz 时钟输入
iCLK_50	PIN_AD15	50MHz 时钟输入
iCLK_50_2	PIN_D16	50MHz 时钟输入
iCLK_50_3	PIN_R28	50MHz 时钟输入
iCLK_50_4	PIN_R3	50MHz 时钟输入
iEXT_CLOCK	PIN_R29	外部时钟(SMAII)输入

将此时钟信号作为计数器的时钟信号，即可构成一个如下计时时间的计时器：

计时时间＝脉冲个数×脉冲周期

4.4.2 实验内容

4.4.2.1　十进制计数器

请设计一个十进制的计数器，计数器值每 1 秒增加一次，并将计数器的值显示在七段数码管上，用 KEY0 作为计数器的清零端。

设计电路，编译工程，用 RTL Viewer 工具查看代码生成的门级电路，然后用 Technology Map Viewer 工具查看计数器在 FPGA 中的实现。对电路进行仿真并下载到 DE2-70 开发板上，验证其功能。

4.4.2.2　时钟的设计

请在 DE2-70 开发板上实现一个时钟，七段数码管分别用于显示时、分和秒。要求此时钟带有预置时间功能。

编译工程，用 RTL Viewer 工具查看代码生成的门级电路，然后用 Technology Map Viewer 工具查看时钟在 FPGA 中的实现。配置引脚，对电路进行仿真并下载到 DE2-70 开发板上，验证其功能。

4.5 存储器实验

存储器(Memory)是计算机系统中的记忆设备,用来存放程序和数据。计算机中全部信息,包括输入的原始数据、计算机程序、中间运行结果和最终运行结果都保存在存储器中。

本实验的目的是了解 FPGA 的片上存储器的存储器特性,分析存储器的工作时序,并学习如何使用 FPGA 的片上存储器。

4.5.1 DE2-70 实验平台上的 M4K

EP2C70 FPGA 片内含有多列 M4K 存储器,每一个 M4K 存储器块含有 4608b(4096 个数据位和 512 个奇偶校验位)。M4K 存储器块内含有同步写入的输入寄存器和输出寄存器。M4K 存储器的输出寄存器可以被旁路,但是输入寄存器不能被旁路。每个 M4K 块可以有不同的配置方法,包括真双口 RAM、简单双口 RAM、单口 RAM、ROM 或者 FIFO 缓存等。

在 Verilog 代码中,可以用多维数组定义存储器。一个 32 字节的 8 位存储器块,可以定义为 32×8 的数组,在 Verilog 语言中可以作如下变量声明:

```
Reg[7:0] memory_array [31:0];
```

在 Cyclone Ⅱ 系列 FPGA 中,这组数组可以由每个逻辑单元中的触发器构建,也可以由 M4K 存储器块实现。想要保证用 M4K 存储器块实现 RAM,必须采用满足 Verilog 代码风格的语言形式来定义 RAM,这样 Quartus Ⅱ 软件在编译的时候,会自动推断出该代码描述的是一个 RAM 块,从而用 M4K 实现 RAM。最简单的方法就是用 Quartus Ⅱ 提供的模板来完成设计。

在 Verilog 语言编辑窗口空白处单击鼠标右键,选择 Insert Template 即可进入模板选择,选择合适的 RAM 模板,插入编辑,如图 4-27 所示。

4.5.2 单时钟简单双口 RAM

利用 Quartus Ⅱ 编译器提供的 Verilog HDL 代码模板,插入单时钟简单双口 RAM 代码模块,如图 4-28 所示。从代码中可以看出,只有在时钟信号有效时(这里是上升沿),才对存储器进行输入和输出操作,这样在综合时,输入端和输出端都会综合出锁存器。

如图 4-28 所示的代码可以综合出一个带有输出寄存器的单时钟简单双口 RAM,图 4-29 是设计代码编写的测试代码,其 RTL Viewer 和功能仿真的结果如图 4-30 和图 4-31 所示。

分析图 4-31 可见,在第 1 个时钟上升沿到来前,写入地址为“0”,写入数据为“FF”,这些数据都稳定在输入地址寄存器和数据寄存器的输入端。读出地址也为“0”,但是,读出地址不经过输入寄存器,直接连到存储模块上,此时存储块输出“0”号地址的内容(初始值无效)稳定在输出寄存器的输入端。第 1 个时钟上升沿到来时,所有输入端的地址和数据

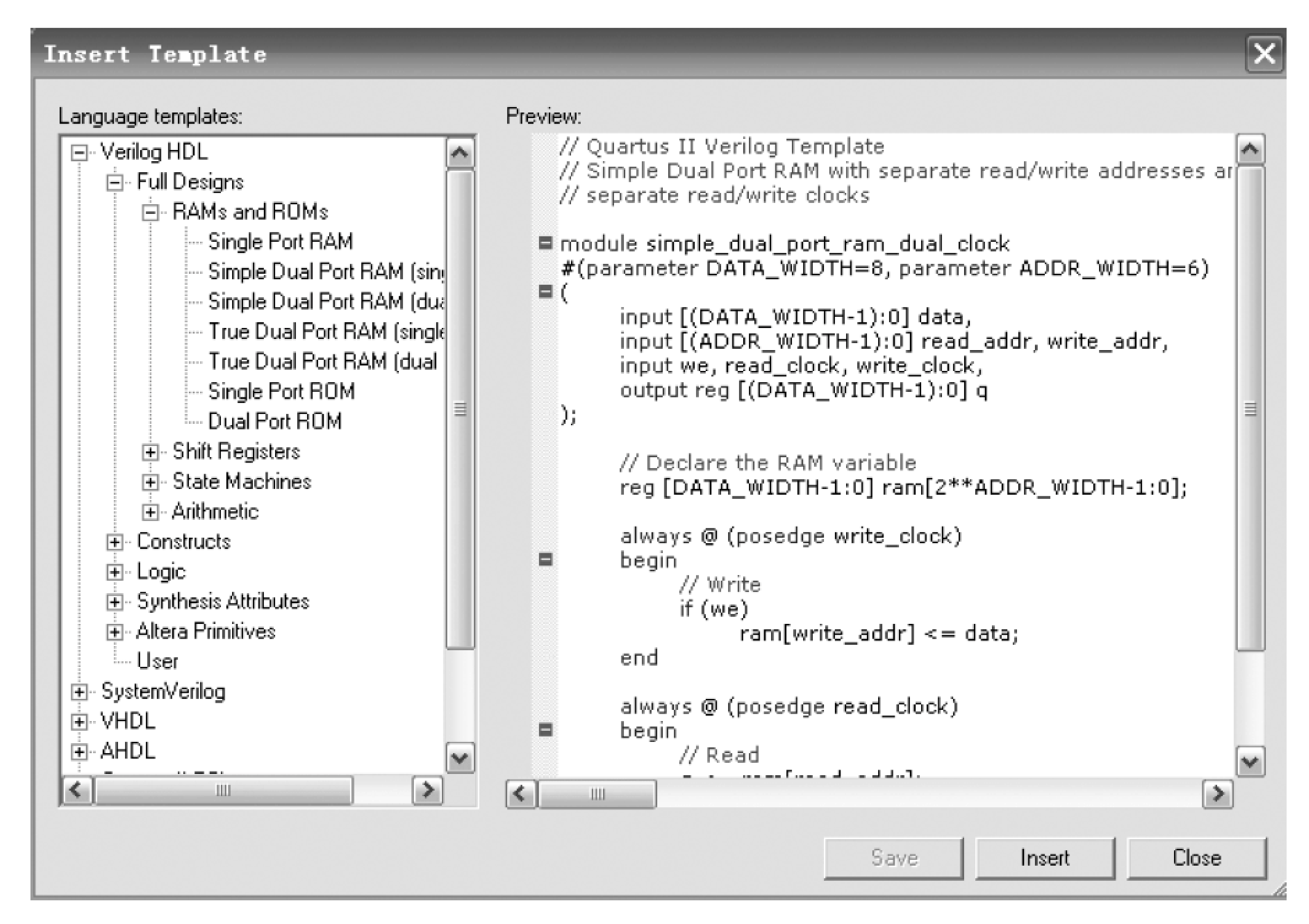


图 4-27 RAM 模板

simple_dual_port_ram_single_clock.v | Compilation Report - Flow Summar

```
// Quartus II Verilog Template
// True Dual Port RAM with single clock


// Quartus II Verilog Template
// Simple Dual Port RAM with separate read/write addresses and
// single read/write clock

module simple_dual_port_ram_single_clock
#(parameter DATA_WIDTH=8, parameter ADDR_WIDTH=6)
(
    input [(DATA_WIDTH-1):0] data,
    input [(ADDR_WIDTH-1):0] read_addr, write_addr,
    input we, clk,
    output reg [(DATA_WIDTH-1):0] q
);

    // Declare the RAM variable
    reg [DATA_WIDTH-1:0] ram[2**ADDR_WIDTH-1:0];

    always @ (posedge clk)
    begin
        // Write
        if (we)
            ram[write_addr] <= data;

        // Read (if read_addr == write_addr, return OLD data).  To return
        // NEW data, use = (blocking write) rather than <= (non-blocking write)
        // in the write assignment.  NOTE: NEW data may require extra bypass
        // logic around the RAM.
        q <= ram[read_addr];
    end

endmodule
```

图 4-28 有输出寄存器的单时钟双口 RAM

```
`timescale 10 ns/ 10 ns
module simple_dual_port_ram_single_clock_vlg_tst();

reg clk;
reg [7:0] data;
reg [5:0] read_addr;
reg we;
reg [5:0] write_addr;
wire [7:0]  q; // wires
simple_dual_port_ram_single_clock i1 (
// port map - connection between master ports and signals/registers
    .clk(clk),
    .data(data),
    .q(q),
    .read_addr(read_addr),
    .we(we),
    .write_addr(write_addr)
);
always
#10 clk = ~clk;

initial
begin
clk = 0; write_addr =6'h00;  data =8'hff;  read_addr =6'h00;  we = 1;
#20      write_addr =6'h02;  data =8'h22;  read_addr =6'h02;
#20      write_addr =6'h10;  data =8'h18;
#20                                        read_addr =6'h00;
#40      write_addr =6'h03;  data =8'h33;  read_addr =6'h03;  we = 0;
#20      write_addr =6'h00;  data =8'h00;  read_addr =6'h03;
#20
$stop;
end
endmodule
```

图 4-29　存储器的 Testbench

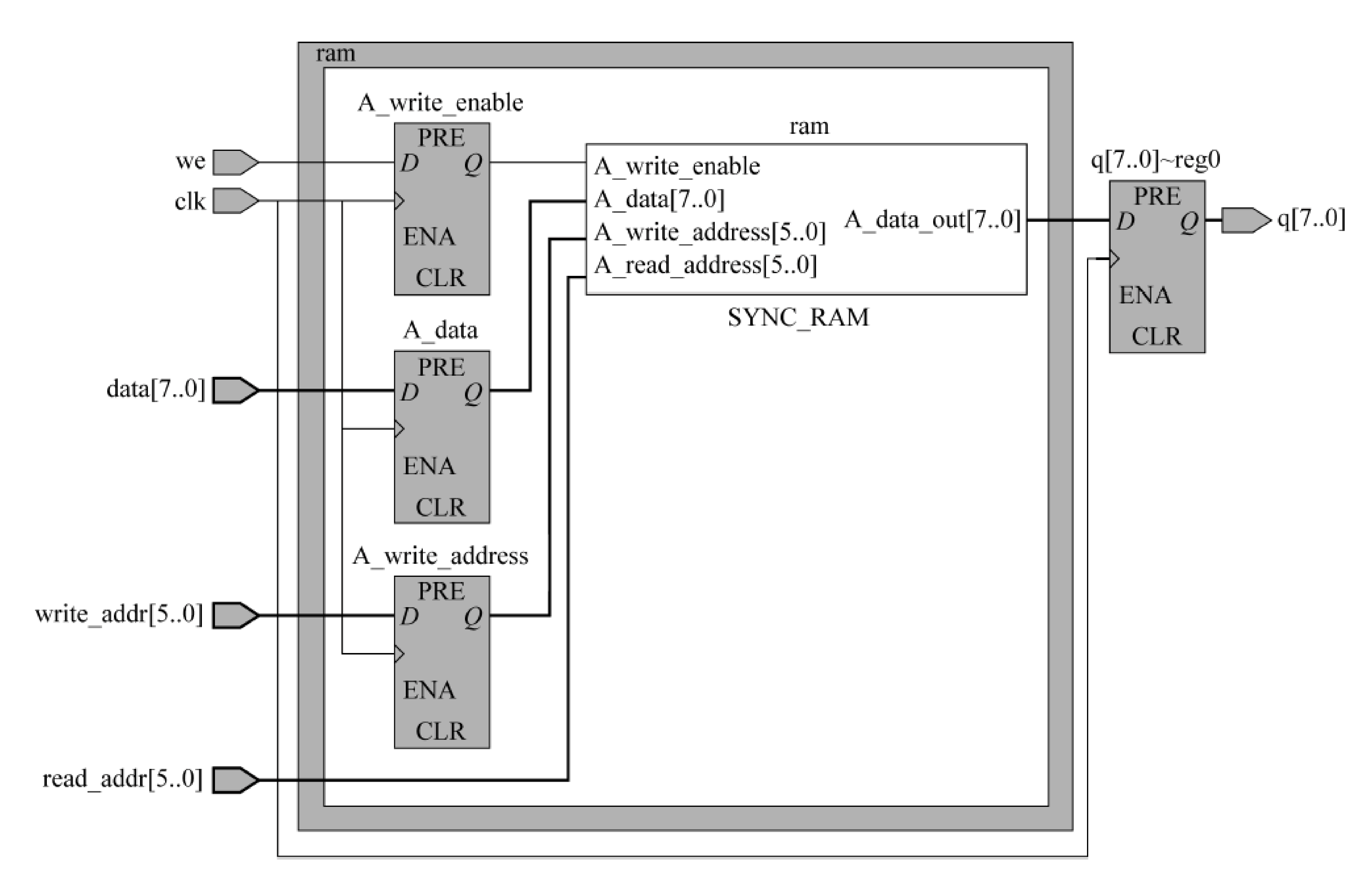


图 4-30　带输出锁存器的 RAM 的 RTL Viewer

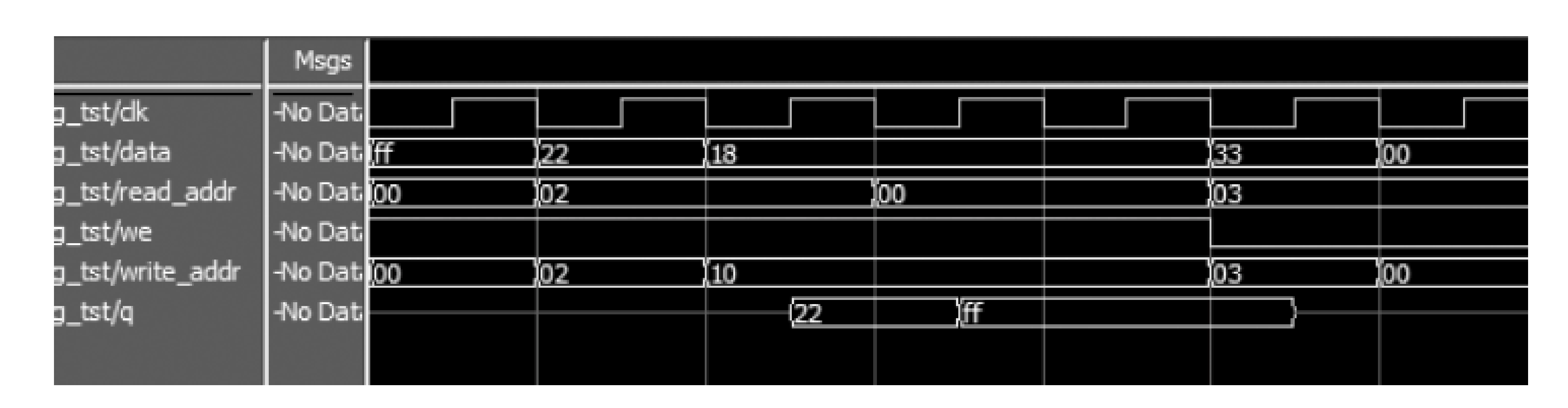


图 4-31　有输出寄存器的单时钟双口 RAM 仿真图

都被写入寄存器中，由于写使能有效，数据被写入存储模块，在输出端，先前“0”号地址的值“无效值”也被输出到了 q 端，图中应为红色线表示。

第 2 个时钟上升沿到来前，输入地址和输出地址都为“2”，输入数据为“22”，当第 2 个时钟上升沿到来时，写使能有效，数据“22”被写入“2”号单元，原来“2”号单元的值(初始值无效值)被输出到输出端 q。第 3 个时钟上升沿到来时，输出地址仍然为“2”，则“2”号单元新的值“22”被输出到输出端 q 上。

第 6 个时钟上升沿到来时，虽然输入地址和输入数据都有效，但是写使能无效，数据不能被写入存储模块，所以在第 7 个时钟上升沿到来时，输出端 q 的值仍然为“3”号存储单元的初始值“无效”。

存储器中的输出寄存器是可以被屏蔽掉的，这只需稍微修改一下代码。代码清单 4.4 就是一个没有输出寄存器的单时钟简单双口 RAM。

程序清单 4.4 输出无锁存的简单单时钟双口 RAM。

```
module simple_dual_port_ram_single_clock
# (parameter DATA_WIDTH= 8, parameter ADDR_WIDTH= 6)
(
    input [(DATA_WIDTH- 1):0] data,
    input [(ADDR_WIDTH- 1):0] read_addr, write_addr,
    input we, clk,
    output  [(DATA_WIDTH- 1):0] q
);
    //Declare the RAM variable
    reg [DATA_WIDTH- 1:0] ram[2**ADDR_WIDTH- 1:0];
    always @ (posedge clk)
    begin
        //Write
        if (we)
            ram[write_addr]<= data;
    end
        //Read
    assign q= ram[read_addr];
endmodule
```

其 RTL Viewer 和功能仿真图分别如图 4-32 和图 4-33 所示。请比较图 4-31 和图 4-33 看看有何区别，你能知道为什么吗？

4.5.3 实验内容

用 LPM 实现 RAM。

Quartus Ⅱ 为常用的逻辑电路，例如加法器、计数器、寄存器以及存储器等，提供了

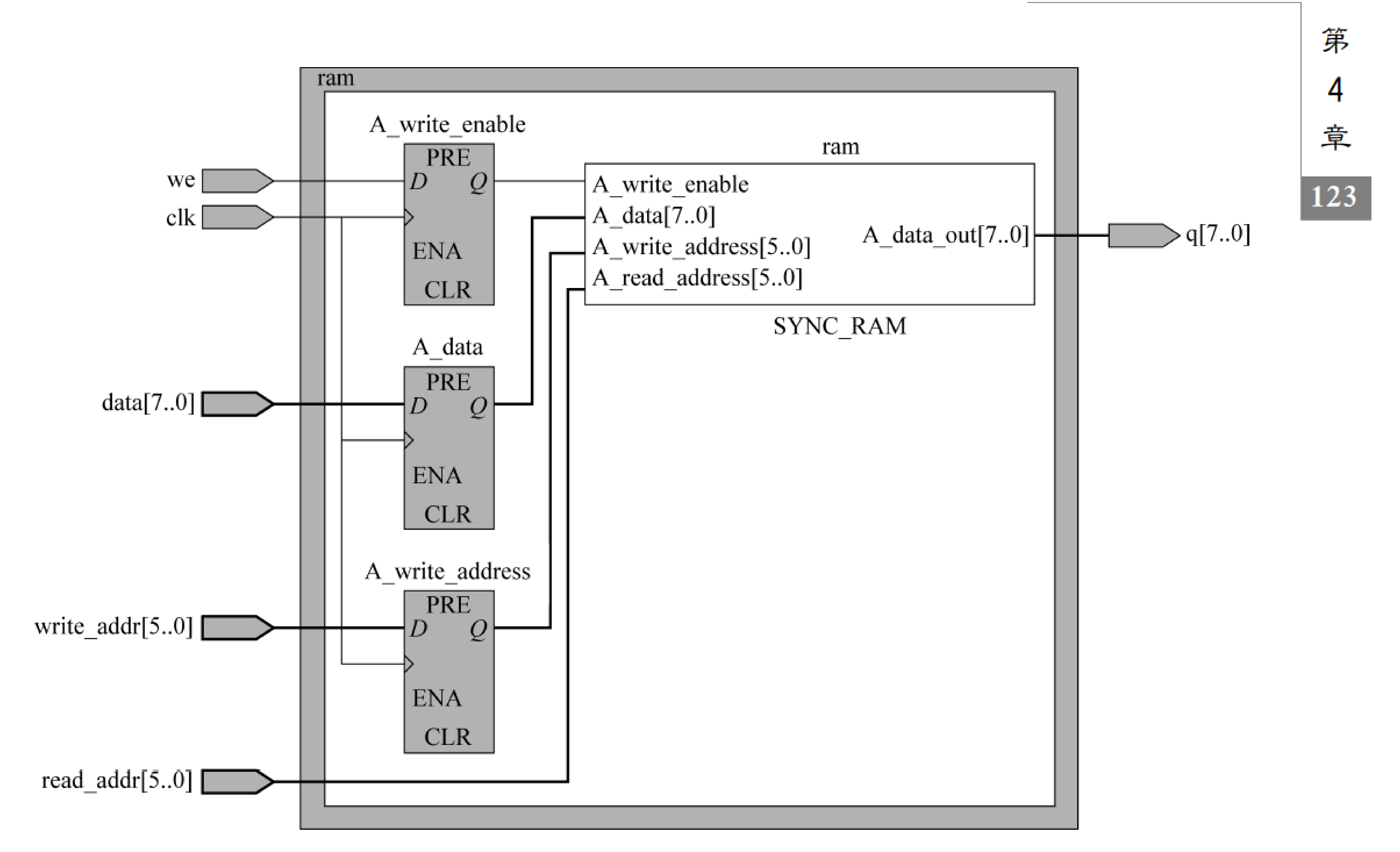


图 4-32　无输出锁存器的 RAM 的 RTL Viewer

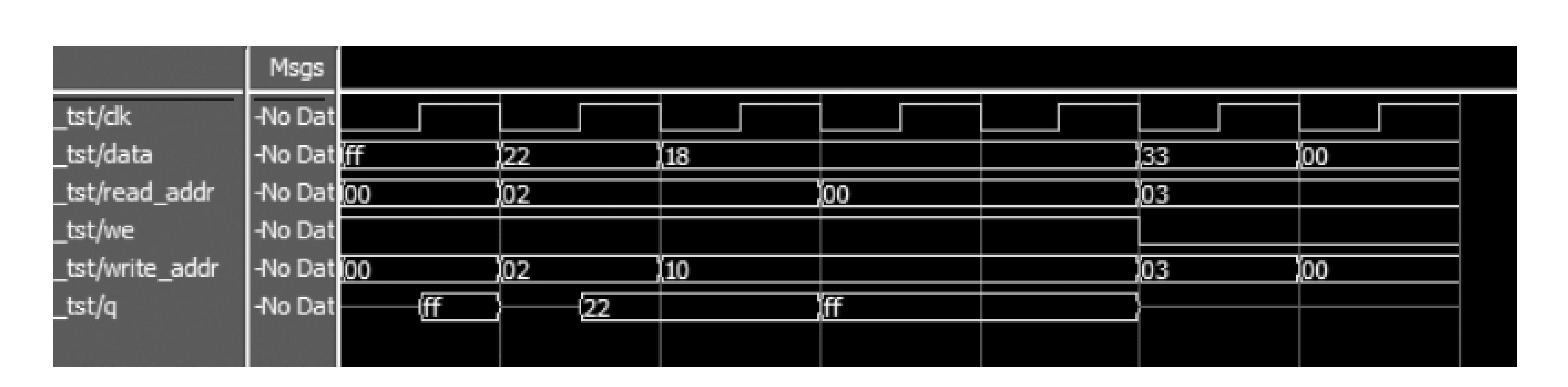


图 4-33　无输出寄存器的单时钟双口 RAM 仿真图

参数化功能模块,用参数化功能模块可以简单地构建我们想要的存储器。请参照加法器实验中构建参数化功能模块的构建方法,构建一个存储器。请详细阅读在构建存储器过程中的参数说明,选择合适的参数构建一个存储器,分析与综合后查看其 RTL Viewer,并对其进行功能仿真,分析仿真图,判断结果是否满足你的设计要求。

第 5 章 状态机和简单数字系统设计

5.1 状态机实验

有限状态机(Finite State Machine,FSM)简称状态机,是一个在有限个状态之间进行转换和动作的计算模型。有限状态机含有一个起始状态、一个输入列表(列表中包含了所有可能的输入信号序列)、一个状态转移函数和一个输出端,状态机在工作时由状态转移函数根据当前状态和输入信号来确定下一个状态和输出。状态机一般都是从起始状态开始,根据输入信号由状态转移函数决定状态机的下一个状态。

有限状态机是数字电路系统中一种十分重要的电路模块,是一种输出取决于过去输入和当前输入的时序逻辑电路,是组合逻辑电路和时序逻辑电路的一种组合。其中组合逻辑分为两个部分,一部分是用于产生有限状态机下一个状态的次态逻辑,另一部分是用于产生输出信号的输出逻辑。除了输入和输出外,状态机还有一组具有"记忆"功能的寄存器,这些寄存器的功能是记忆有限状态机的内部状态,常被称为状态寄存器。

本实验的目的是学习状态机的工作原理,了解状态机的编码方式,并学习简单状态机的设计。

5.1.1 有限状态机

在实际应用中,有限状态机被分为两种:Moore 型有限状态机和 Mealy 型有限状态机。

Moore 型有限状态机的输出信号只与有限状态机的当前状态有关,与输入信号的当前值无关,输入信号的当前值只会影响到状态机的次态,不会影响到状态机当前的输出。即 Moore 型有限状态机的输出信号是直接由状态寄存器译码得到。Moore 型有限状态机在时钟 CLK 信号有效后再经过一段时间的延迟,输出达到稳定值。即使在这个时钟周期内输入信号发生变化,输出也会在这个完整的时钟周期内保持稳定值而不再变化。输入对输出的影响要到下一个时钟周期才能反映出来。Moore 有限状态机最重要的特点就是将输入与输出信号隔离开来。

Mealy 状态机与 Moore 有限状态机不同,Mealy 有限状态机的输出不仅与状态机的当前状态有关,而且与输入信号的当前值也有关。Mealy 有限状态机的输出直接受输入

信号的当前值影响，而输入信号可能在一个时钟周期内任意时刻发生变化，这使得 Mealy 有限状态机对输入的响应发生在当前时钟周期，比 Moore 有限状态机对输入信号的响应要早一个周期。因此，输入信号的噪声可能影响到输出的信号。

5.1.2 简单状态机 FSM

本节通过设计一个实际的状态机来了解状态机的工作过程和设计方法。

请设计一个区别两种特定时序的有限状态机 FSM：该有限状态机有一个输入 w 和一个输出 z。当 w 是 4 个连续的“0”或者 4 个连续的“1”时，输出 $z=1$，否则 $z=0$。时序允许重叠，若 w 是连续的 5 个“1”时，则在第 4 个和第 5 个时钟之后，z 均为“1”。图 5-1 是这个有限状态机的时序图。

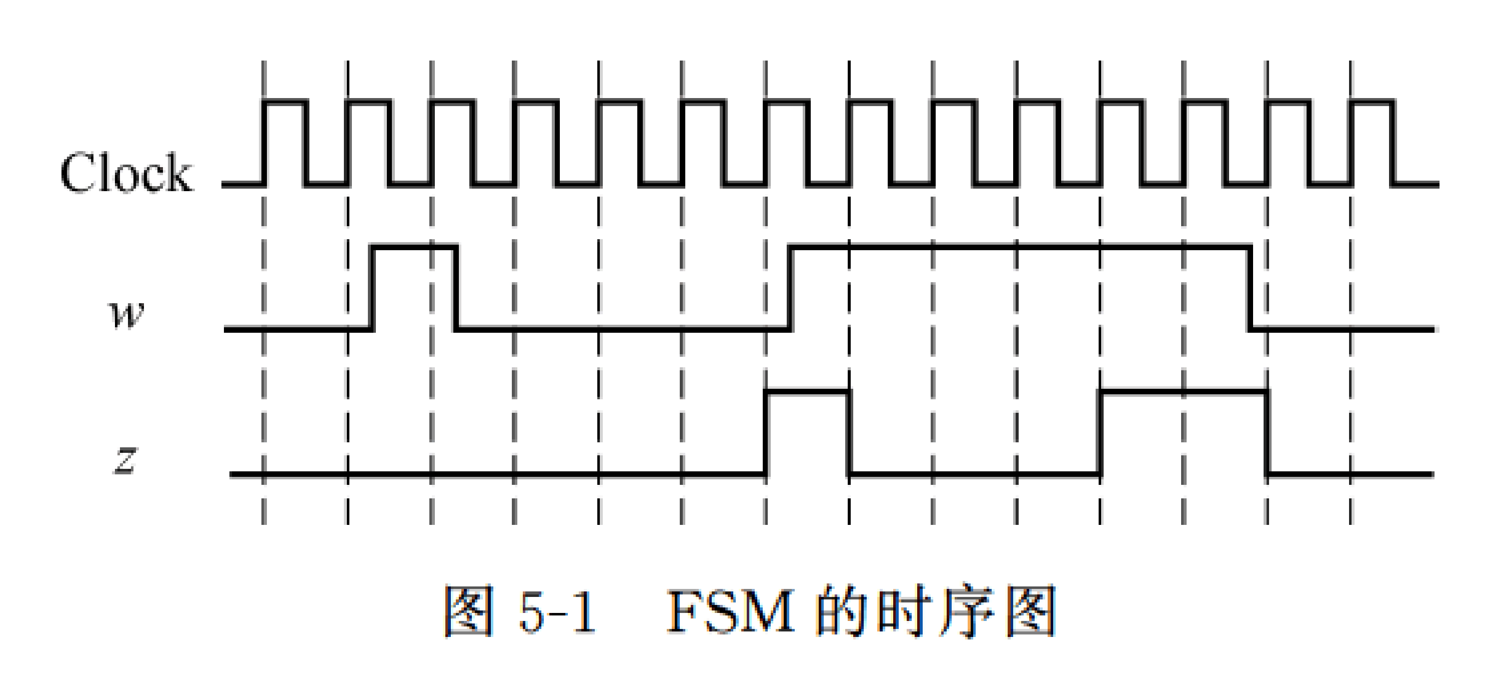


图 5-1　FSM 的时序图

这个状态机的状态图如图 5-2 所示。

用 Verilog 代码实现如图 5-2 所示的状态机。此状态机有 9 个状态，至少需要 4 个状态寄存器按照二进制编码形式实现这个 FSM，如表 5-1 所示。

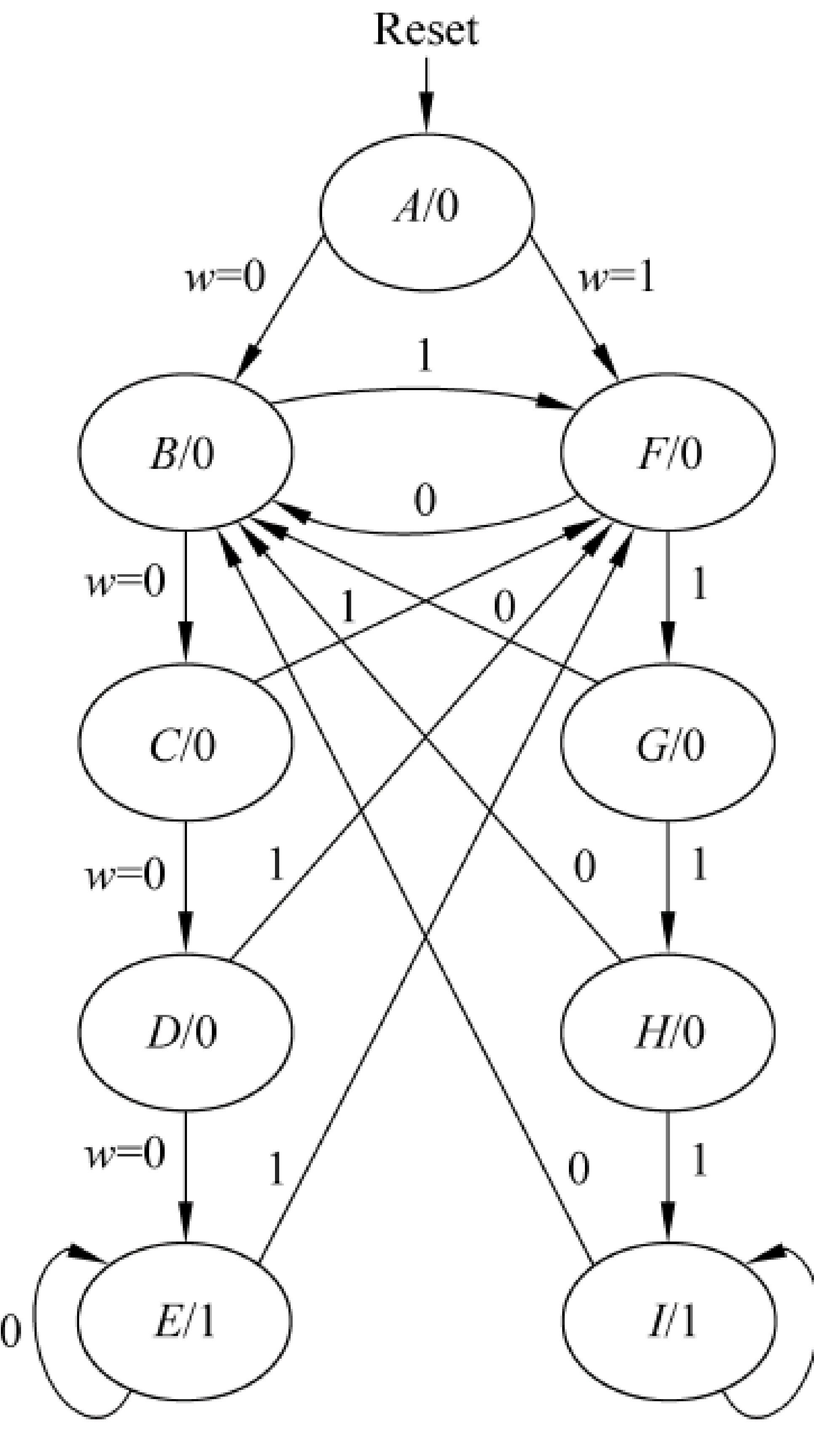


图 5-2　FSM 的状态图

表 5-1　FSM 的二进制编码

状态	y_3	y_2	y_1	y_0
A	0	0	0	0
B	0	0	0	1
C	0	0	1	0
D	0	0	1	1
E	0	1	0	0
F	0	1	0	1
G	0	1	1	0
H	0	1	1	1
I	1	0	0	0

建立一个 Verilog 文件，用 iSW[0]作为 FSM 低电平有效同步复位端，用 iSW[1]作为输入 w，用 iKEY[0]作为手动的时钟输入，用 oLEDG[0]作为输出 z，用 oLEDR[3]～oLEDR[0]显示 9 个状态，完成该状态机的具体设计。

Quartus Ⅱ里面自带了用 HDL 语言实现的状态机模板，在 Quartus Ⅱ的文件编辑窗口单击右键选择“insert template”，参考 template 可以提高状态机设计效率，如图 5-3 和图 5-4 所示。程序清单 5.1 即为此状态机的 Verilog HDI 代码。

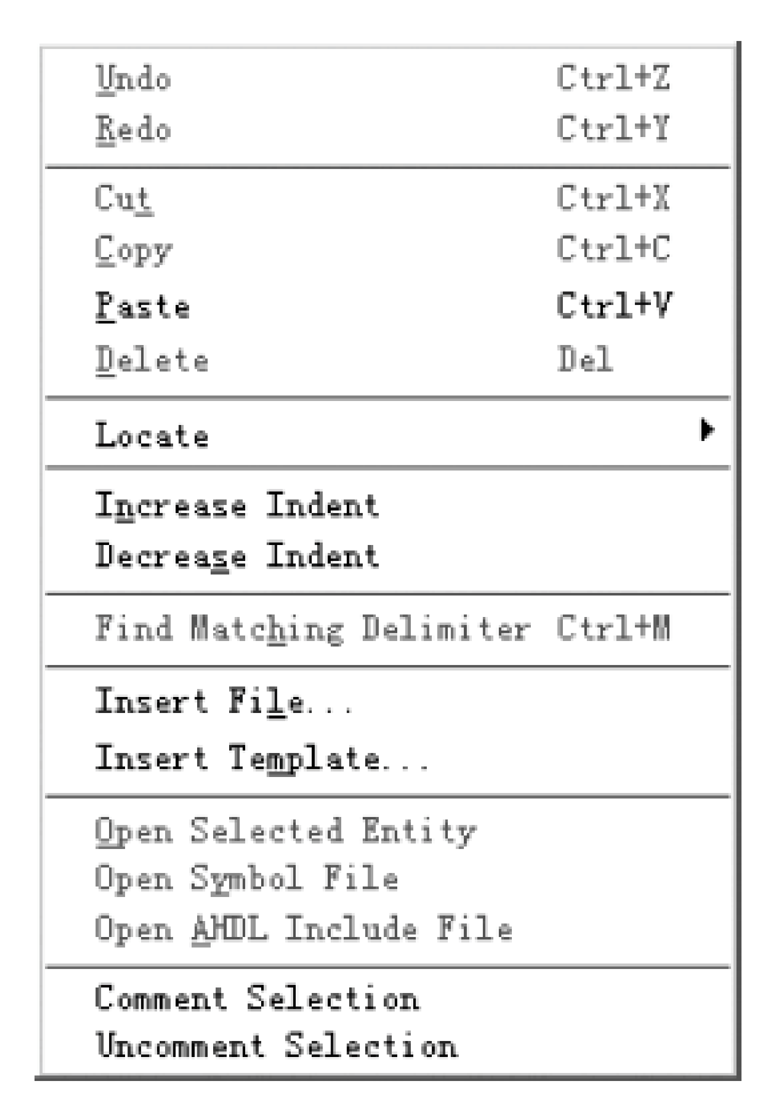


图 5-3　选择 Quartus Ⅱ设计模板

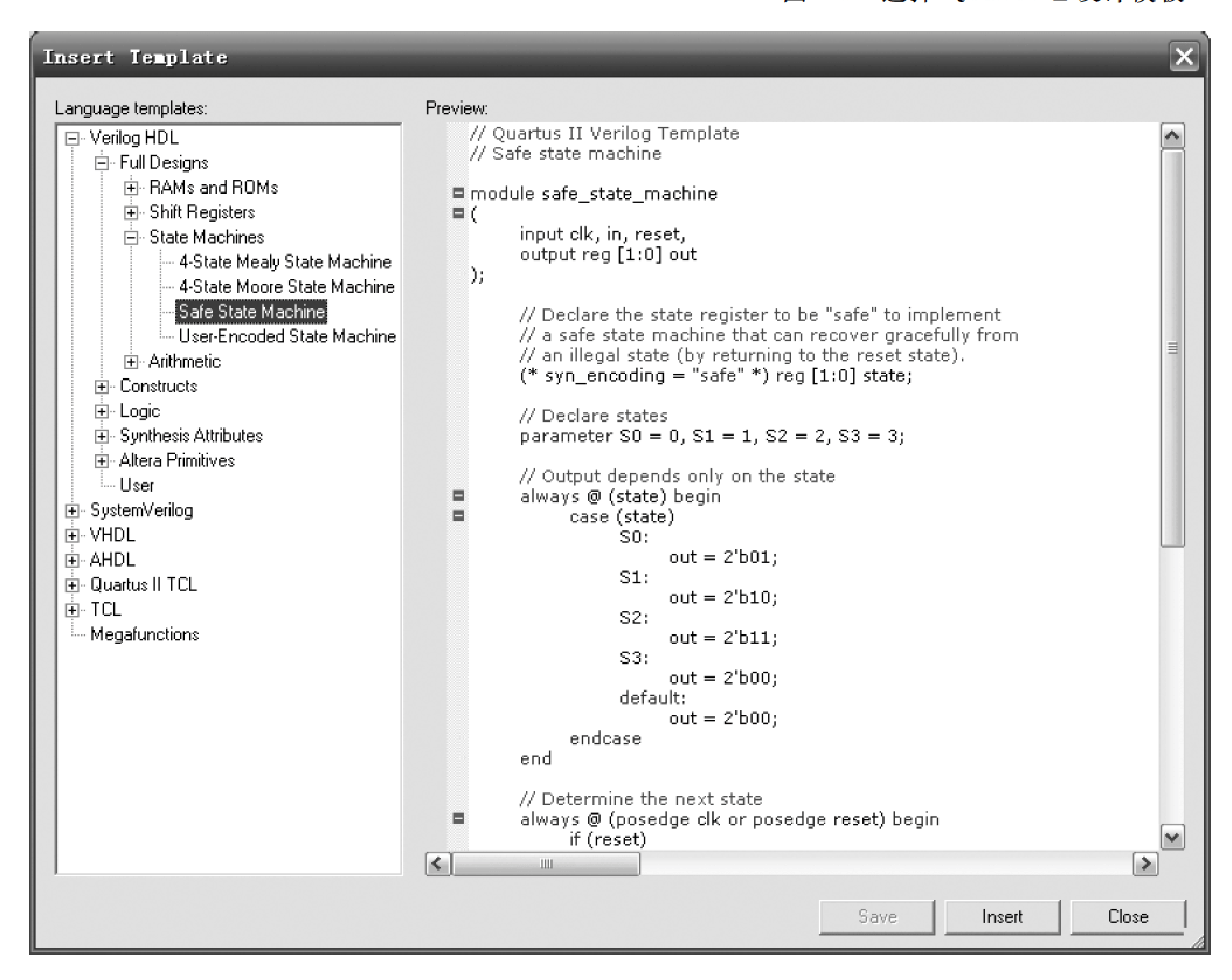


图 5-4　模板代码

程序清单 5.1　区别两种输入状态的状态机。

```
module FSM_bin
(
    input    clk, in, reset,
    output reg out
);
```

```
    //Declare the state register to be "safe" to implement
    //a safe state machine that can recover gracefully from
    //an illegal state (by returning to the reset state).
    (* syn_encoding="safe" *) reg [3:0] state;
    //Declare states
    parameter S0=0, S1=1, S2=2, S3=3, S4=4, S5=5, S6=6, S7=7, S8=8;
    //Output depends only on the state
    always @ (state) begin
        case (state)
            S0: out=1'b0;
            S1: out=1'b0;
            S2: out=1'b0;
            S3: out=1'b0;
            S4: out=1'b1;
            S5: out=1'b0;
            S6: out=1'b0;
            S7: out=1'b0;
            S8: out=1'b1;
            default: out=1'bx;
        endcase
    end
    //Determine the next state
    always @ (posedge clk) begin
        if (reset)          state<=S0;
        else
            case (state)
                S0: if (in)     state<=S5;    else    state<=S1;
                S1: if (in)     state<=S5;    else    state<=S2;
                S2: if (in)     state<=S5;    else    state<=S3;
                S3: if (in)     state<=S5;    else    state<=S4;
                S4: if (in)     state<=S5;    else    state<=S4;
                S5: if (in)     state<=S6;    else    state<=S1;
                S6: if (in)     state<=S7;    else    state<=S1;
                S7: if (in)     state<=S8;    else    state<=S1;
                S8: if (in)     state<=S8;    else    state<=S1;
            endcase
    end
endmodule
```

编译工程，用 Netlist Viewers 工具查看代码生成的状态图，如图 5-5 所示。如果编译

结果不能产生如图 5-5 所示的状态图，可能是用户编写的代码不符合 Quartus Ⅱ 编译器要求的状态机代码风格，重新采用 Quartus Ⅱ 编译器能够识别的方式编写状态机，就可以产生状态图。

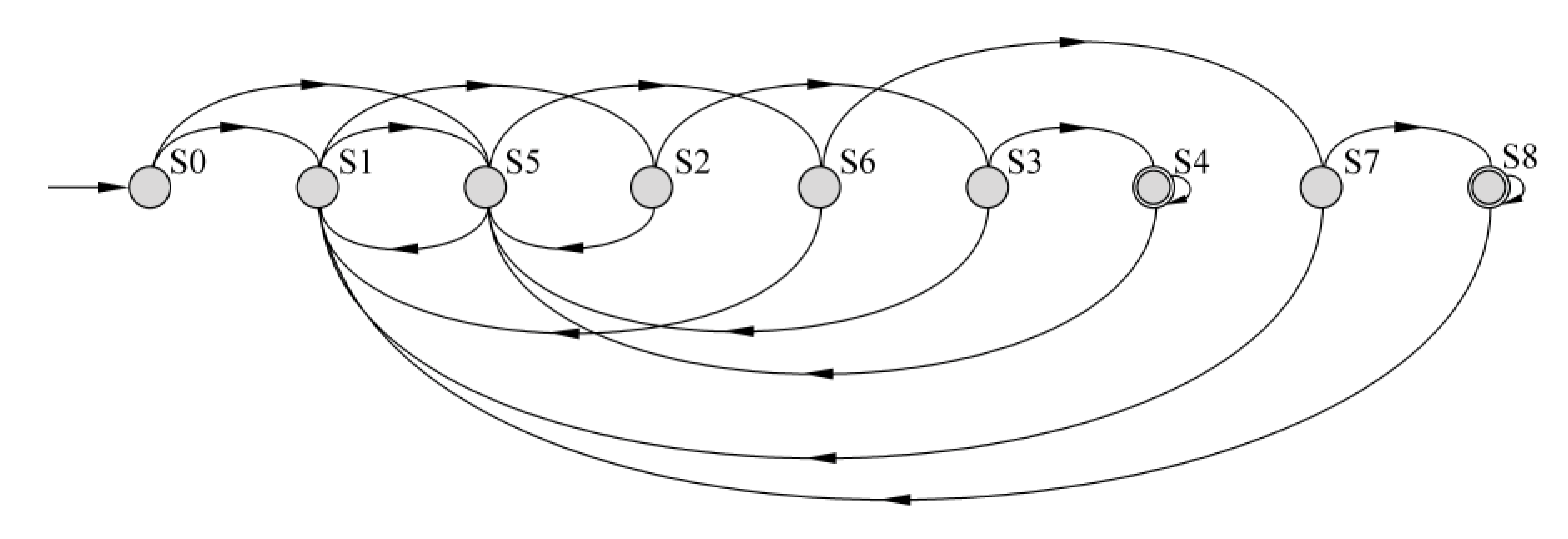


图 5-5　生成状态图

为此状态机编写测试代码，如图 5-6 所示。图 5-7 是此状态机的功能仿真结果。

```
`timescale 10 ns/ 1 ps
module FSM_bin_vlg_tst();
// constants
// test vector input registers
reg clk;
reg in;
reg reset;
// wires
wire out;

FSM_bin i1 (
// port map - connection between master ports and signals/registers
   .clk(clk),
   .in(in),
   .out(out),
   .reset(reset)
);
initial
begin
clk = 0; in = 0; reset = 1;
#15 in = 1; reset = 0;
#20 in = 0;
#40 in = 1;
#150 reset = 1;
#20 in = 0;
#50;
$stop;
end
always
#10 clk = ~clk;
endmodule
```

图 5-6　测试代码

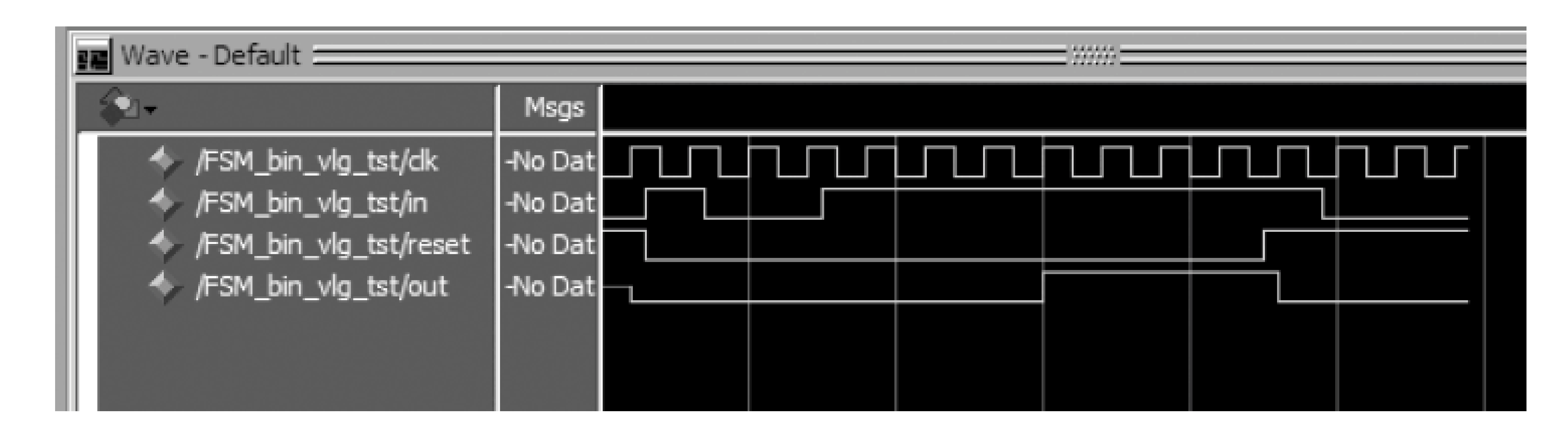


图 5-7　功能仿真

用 RTL Viewer 工具查看代码生成的门级电路，然后用 Technology Map Viewer 工具查看状态机在 FPGA 中的实现。对电路进行仿真并下载到 DE2-70 开发板上，验证其

功能。

5.1.3 状态机的编码方式

上一节例子中的状态机采用顺序二进制编码 binary 方式，即将状态机的状态依次编码为顺序的二进制数，用顺序二进制数编码可使状态向量的位数最少。如本例中只需要4位二进制数来编码。节省了保存状态向量的逻辑资源。但是在输出时要对状态向量进行解码以产生输出(某个状态有特定的输出，因此输出时要对此状态进行解码，满足状态编号时才可输出特定值)，这个解码过程往往需要许多组合逻辑。

另外，当芯片受到辐射或者其他干扰时，可能会造成状态机跳转失常，甚至跳转到无效的编码状态而出现死机。例如，状态机因为异常跳转到某状态，而此状态需要等待输入，并作出应答，此时因为状态运转不正常，不会出现输入，状态机就会进入死等状态。

one-hot 编码也是状态机设计中常用的编码，在 one-hot 编码中，对于任何给定的状态，其状态向量中只有 1 位是“1”，其他所有位的状态都为“0”，n 个状态就需要 n 位的状态向量，所以 one-hot 编码最长。one-hot 编码对于状态的判断非常方便，如果某位为“1”就是某状态，如果该位为“0”则不是此状态。因此，判断状态输出时非常简单，只要一、两个简单的“与门”或者“或门”即可。

one-hot 编码的状态机从一个状态到另一个状态的状态跳转速度非常快，而顺序二进制编码从一个状态跳转到另外一个状态则需要较多次跳转，并且随着状态的增加，速度急剧下降。

在芯片受到干扰时，one-hot 编码一般只能跳转到无效状态(如果跳到另一有效状态必须是当前为“1”的变为“0”，同时另外一位由“0”变为“1”，这种可能性很小)，因此 one-hot 编码的状态机稳定性高。

格雷码(gray-code)也是状态机设计中常用一种编码方式，它的优点是 gray-code 状态机在发生状态跳转时，状态向量只有 1 位发生变化。

一般而言，顺序二进制编码和 gray-code 的状态机使用了最少的触发器，较多的组合逻辑，适用于提供更多的组合逻辑的 CPLD 芯片。对于具有更多触发器资源的 FPGA，用 one-hot 编码实现状态机则更加有效。所以，CPLD 多使用 gray-code，而 FPGA 多使用 one-hot 编码。另一方面，对于小型设计使用 gray-code 和 binary 编码更有效，而大型状态机设计使用 one-hot 更高效。

在表 5-2 中，状态机有 9 个状态，我们可以用 9 个触发器来实现这个状态机的电路，这个状态机的电路就是从输入到每一个状态触发器输出端的逻辑表达式的实现电路。这 9 个状态触发器用 $y_8\ y_7\ y_6\ y_5\ y_4\ y_3\ y_2\ y_1\ y_0$ 表示。该状态机的 one-hot 编码如表 5-2 所示。

5.1.4 实验内容

建立一个 Verilog 文件，调用 9 个触发器来实现这个 FSM，用 iSW[0]作为 FSM 低电平有效同步复位端，用 iSW[1]作为输入 w，用 iKEY[0]作为手动的时钟输入，用 oLEDG[0]作为输出 z，用 oLEDR[8]～oLEDR[0]显示 9 个触发器的状态。

表 5-2 FSM 的 one-hot 编码

状态	y_8	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
A	0	0	0	0	0	0	0	0	1
B	0	0	0	0	0	0	0	1	0
C	0	0	0	0	0	0	1	0	0
D	0	0	0	0	0	1	0	0	0
E	0	0	0	0	1	0	0	0	0
F	0	0	0	1	0	0	0	0	0
G	0	0	1	0	0	0	0	0	0
H	0	1	0	0	0	0	0	0	0
I	1	0	0	0	0	0	0	0	0

编译工程，用 Netlist Viewers 工具查看代码生成的状态图，然后用 Technology Map Viewer 工具查看状态机在 FPGA 中的实现。对电路进行仿真并下载到 DE2-70 开发板上，验证其功能。

请完成该设计，并与上一个二进制编码的 FSM 进行比较。

5.2 雷鸟车尾灯控制器*

5.2.1 实验目的

大部分的控制系统都可以描述为一个有限状态机，复杂的计算机系统其实也是一个有限状态机系统，本次实验通过一个实例，进一步了解状态机的工作过程和设计方法。

5.2.2 实验内容

1965 年生产的福特雷鸟汽车的车尾每边有三个灯，这些灯轮流地按顺序亮起，表示车子的转向。当驾驶员发出将要向左(LEFT)转的指令时，车尾左侧的三个灯按顺序分别亮 0、1 和 2 三个灯。当驾驶员发出将要向右(RIGHT)转的指令时，车尾右侧的三个灯按顺序分别亮 0、1 和 2 三个灯。另外，还有一个应急闪烁输入(HAZ)，它要求车尾灯工作在告警状态，这时所有 6 个灯协调地闪烁。

5.2.3 问题分析

本设计有三个输入端：LEFT、RIGHT 和 HAZ，告警状态优先级最高，有告警信号时首先进入告警状态。汽车在空闲状态时，输入信号要求汽车进入的工作状态如表 5-3 所示。另外，还需要一个单独运行的时钟信号，该信号的频率等于这些灯所要求的闪烁频率。

汽车有 4 个工作状态：空闲状态、左转弯状态、右转弯状态和告警状态，汽车的左转弯状态和右转弯状态又分为 4 个状态：空闲及 0、1 和 2 三个灯亮。因此，本设计用一个

* 本例选自《数字设计原理与实践》第四版中的 7.5 节。

时钟同步 Moore 状态机来实现，这个状态机有 8 个状态，每个状态对应的输出如表 5-4 所示。

表 5-3　汽车工作状态表

HAZ	LEFT	RIGHT	汽车工作状态
0	0	0	空闲
0	1	0	左转弯
0	0	1	右转弯
0	1	1	告警
1	×	×	告警

表 5-4　工作状态对应的输出

状　　态	输　　出
IDLE	空闲状态，6 个灯全灭
L_1	左边 1 灯亮
L_2	左边 2 灯亮
L_3	左边 3 灯亮
R_1	右边 1 灯亮
R_2	右边 2 灯亮
R_3	右边 3 灯亮
ERR	告警状态，6 个灯全亮

状态转换约定：在进入左转弯状态时，状态机在 IDLE、L_1、L_2 和 L_3 4 个状态间循环，此时只有在 IDLE 状态可以接收任何其他信号输入，并选择进入相应状态。在 L_1 和 L_2 状态只能接收告警状态输入，如果有告警信号，结束左转弯循环，进入告警状态。进入 L_3 状态后不接受任何信号输入，直接进入 IDLE 状态。右转弯状态类似左转弯状态。告警状态在 ERR 状态和 IDLE 状态循环转换。没有任何信号输入时，状态机保持在 IDLE 状态。

状态图如图 5-8 所示。

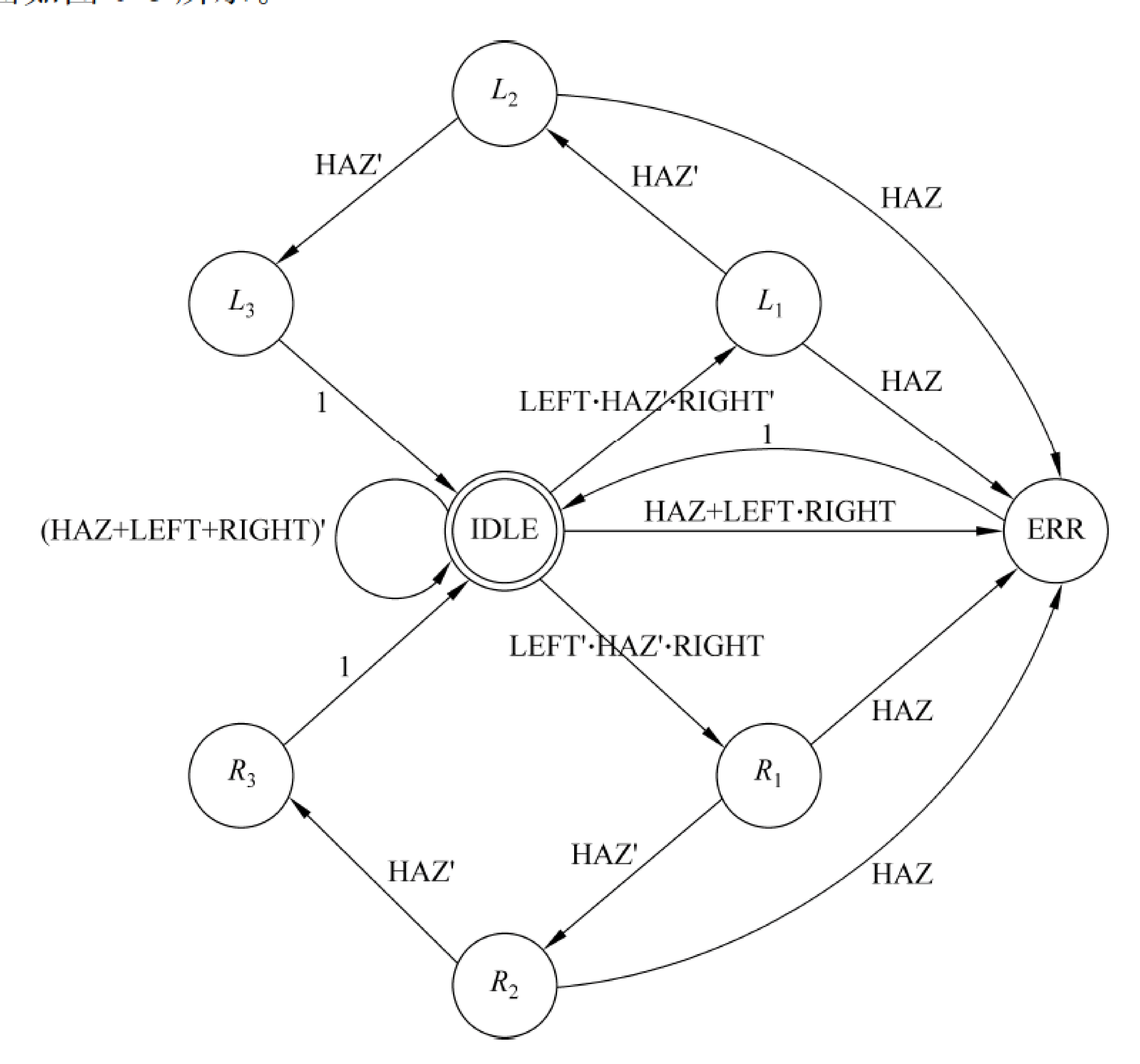


图 5-8　雷鸟车尾灯状态图

请根据上述分析，设计出雷鸟汽车尾灯控制器。并利用 FPGA 开发板资源，验证你设计的正确性。

5.3 交通控制灯实验

5.3.1 实验目的

这是一个综合性的实验，目的是复习巩固以前学过的方法和知识，请读者通过自己的努力设计一个相对复杂的、有实际用途的数字控制系统，学习应用已经学到的知识，了解“我们生活在数字世界中”这一概念。

5.3.2 实验内容

要求设计一个交通信号灯控制电路，主要适用于在两条干道（一条主干道一条支干道）汇合点形成的十字交叉路口，路口设计两组红绿灯分别对两个方向上的交通运行状态进行管理。实验要求如下：

（1）能显示十字路口东西、南北两个方向的红、黄、绿的指示状态。用两组红、黄、绿三色灯作为两个方向的红、黄、绿灯。（注：本实验板上没有黄灯，请想办法用其他方式代替。）

（2）能实现正常的倒计时功能。用两组数码管作为东西和南北方向的倒计时显示，主干道每次放行（绿灯）60 秒，支干道每次放行（绿灯）45 秒，在每次由绿灯变成红灯的转换过程中，要亮黄灯 5 秒作为过渡。

（3）能实现总体清零功能。按下该键后，系统实现总清零，计数器由初始状态计数，对应状态的指示灯亮。

（4）可变换亮灯时间（选做）。由于夜间 21:00 点至 05:00 点之间路上车辆很少，主干道和支干道相比放行时间太长，因此一些交通路口设置了不同时间段的放行时间，请修改你的设计，优化交通灯功能。这里的时间控制最好采用自动的方式，而不是等待外界输入信号而改变的方式，因为设计者不可能在每个夜间人工改变路灯的放行时间。

请根据所学实验方法，完成该设计。

第 6 章 简单接口控制器设计

6.1 PS/2 接口原理及实现

PS/2 是个人计算机串行 I/O 接口的一种标准，因其首次在 IBM PS/2(Personal System/2)机器上使用而得名，PS/2 接口可以连接 PS/2 键盘和 PS/2 鼠标。

本实验的目的是学习 PS/2 键盘接口原理，学习 PS/2 键盘接口控制器的设计方法。

6.1.1 PS/2 接口简介

6.1.1.1 PS/2 接口外观和引脚功能

PS/2 接口是一个圆形的 6 针(或 6 孔)接口，如图 6-1 所示。一个 PS/2 接口可以连接一个输入设备，为了区别起见，在稍旧的计算机中紫色的 PS/2 接口用于接键盘，绿色的 PS/2 接口用于接鼠标，现在的计算机鼠标接口一般都不用 PS/2 接口而改用 USB 接口了。

图 6-2 是 PS/2 接口孔头的引脚示意图，其每个引脚的功能描述如表 6-1 所示。



(a) PS/2接口孔头



(b) PS/2接口针头



图 6-1 PS/2 接口

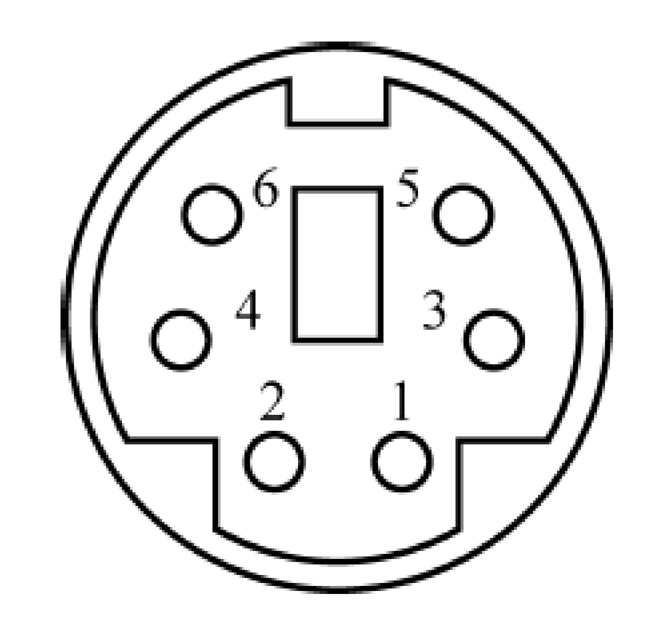


图 6-2 PS/2 接口孔头引脚示意图

6.1.1.2 PS/2 接口的工作时序

键盘处理器在识别到有键按下或放开时都会从 PS2_KBDAT 引脚送出扫描码。当按键被按下时送出的扫描码被称为“通码(Make Code)”；当按键被释放时送出的扫描码

表 6-1　PS/2 接口引脚功能

Pin NO.	Signal Name	Description
1	DATA	PS/2 data
2	NC	Not connected(reserved for second PS/2 device)
3	GND	Ground
4	V_{CC}	+5V DC
5	CLK	Clock
6	NC	Not connected(reserved for second PS/2 device)

称为"断码(Break Code)"。以"W"键为例,"W"键的通码是 1DH,如果"W"键被按下,则 PS2_KBDAT 引脚将输出 1DH,如果"W"键一直没有释放,则不断输出扫描码 1DH,1DH,…,1DH,直到有其他键按下或者"W"键被放开。某按键的断码是 F0H 加此按键的通码,如释放"W"键时输出的断码为 F0H,1DH。

多个键被同时按下时,将逐个输出扫描码。例如,先按左"Shift"键(扫描码为 12H)、再按"W"键、放开"W"键、再放开左"Shift"键,则此过程送出的全部扫描码为 12H,1DH,F0H,1DH,F0H,12H。

键盘和主机间可以进行数据双向地传送,这里只讨论键盘向主机传送数据的情况。当 KBDAT 和 KBCLK 信号线都为高电平(空闲)时,键盘才可以给主机发送信号。如果主机将 KBD Clock 信号置低,键盘将准备接受主机发来的命令。

键盘以每帧 11 位的格式传送数据给主机。第一位是开始位(逻辑"0"),后面跟 8 位数据位(低位在前),1 位奇偶校验位(奇校验)和 1 位停止位(逻辑"1")每位都在时钟的下降沿有效,图 6-3 显示了键盘传送一字节数据的时序。

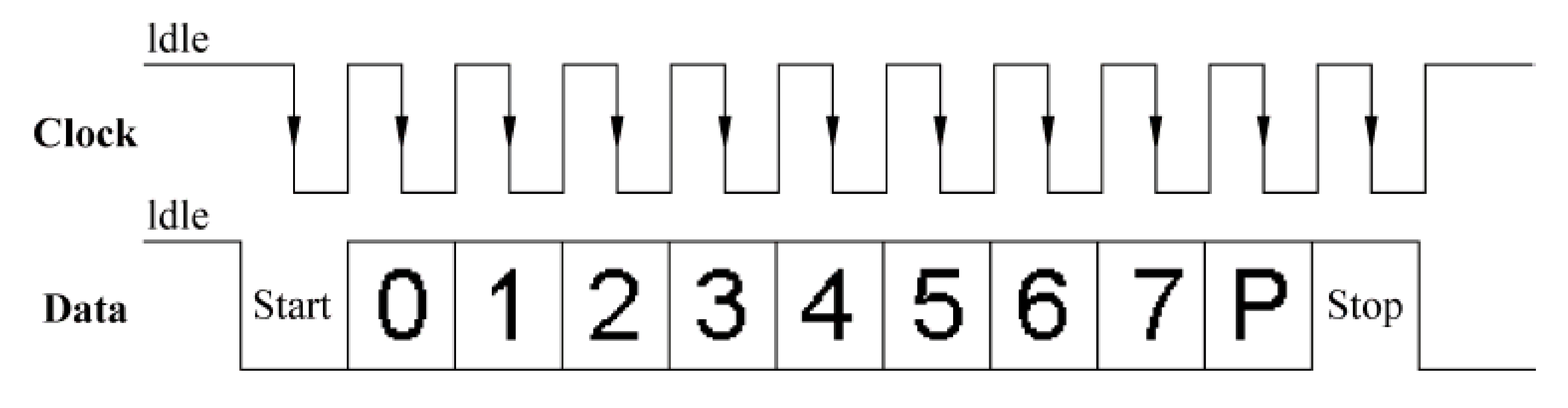


图 6-3　键盘输出数据时序图

6.1.1.3　键盘扫描码

当有键按下、释放或者按住时,键盘处理器都会将此键的扫描码从 Data 端口发送出去,每个键都有唯一的通码和断码。键盘所有键的扫描码组成的集合称为扫描码集,共有三套标准的扫描码集,所有现代的键盘默认使用第二套扫描码。图 6-4 显示了键盘各键的扫描码,例如 Caps 键的扫描码是 58H,扫描码均以十六进制表示。由图 6-4 可以看出,键盘上各按键的扫描码是随机排列的,如果想迅速地将键盘扫描码转换为 ASCII 码,一个最简单的方法就是利用查表的方法,扫描码到 ASCII 码的转换表格请读者自己生成。

图 6-5 是扩展键盘和数字键盘的扫描码。

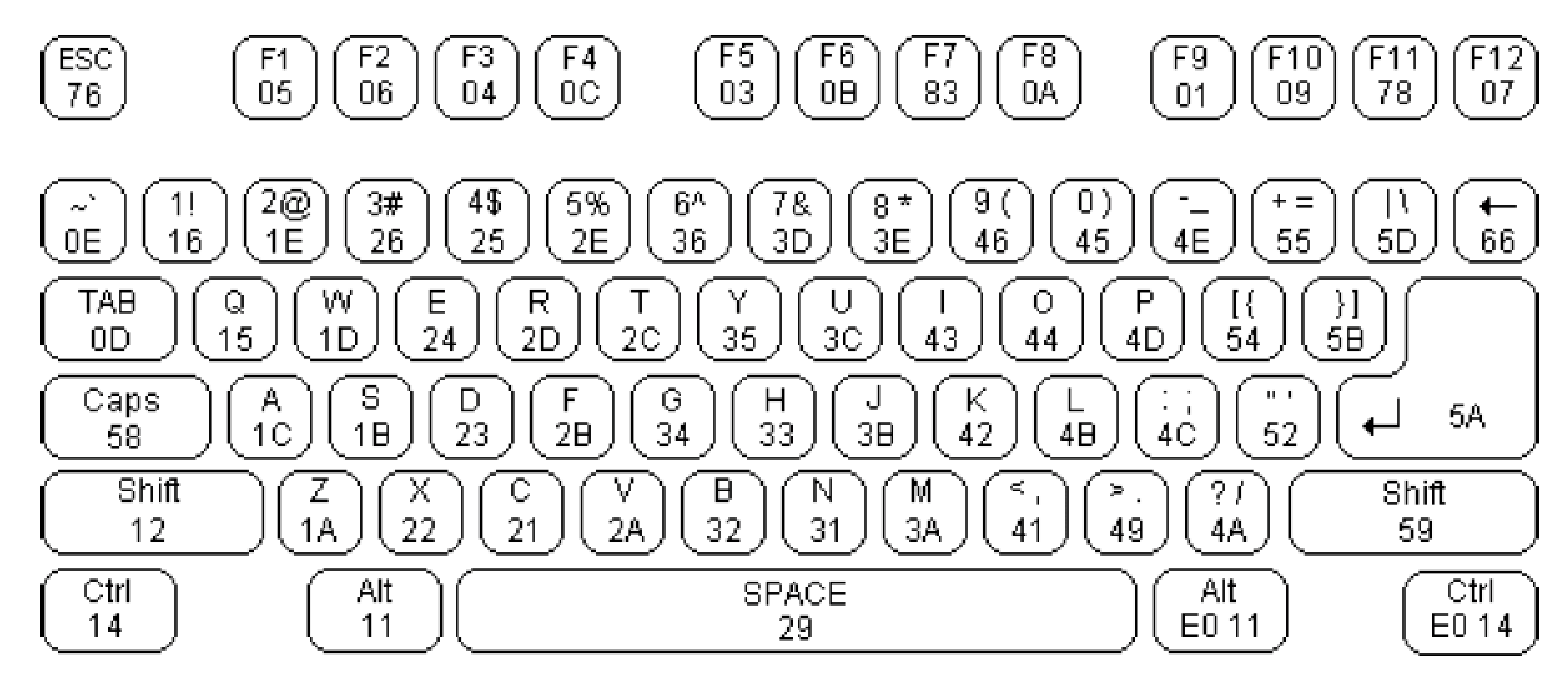


图 6-4 键盘扫描码

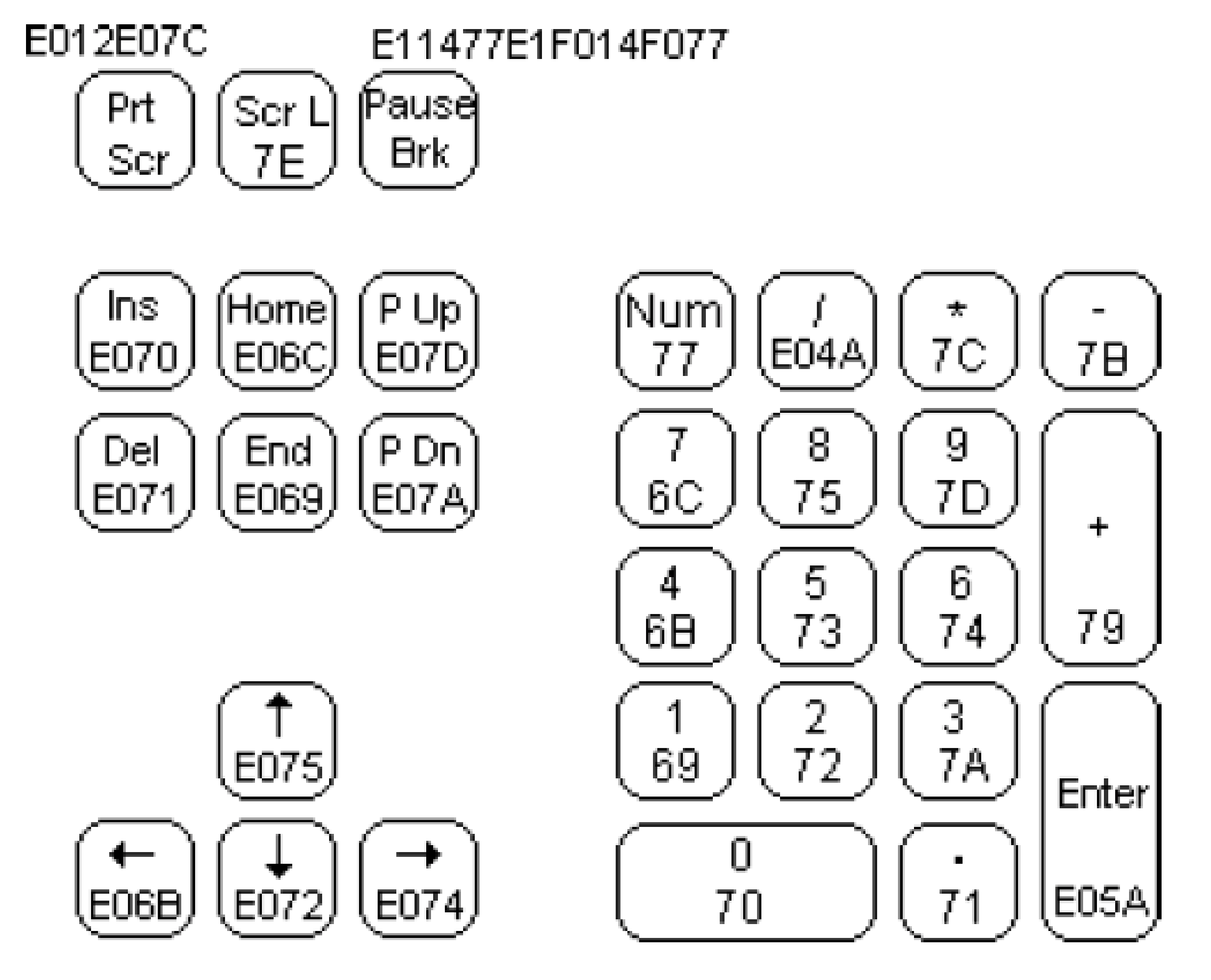


图 6-5 扩展键盘和数字键盘的扫描码

6.1.2 PS/2 接口与 FPGA 的连接

图 6-6 所示的是 DE2-70 开发板上 PS/2 接口的内部电路连接示意图，其中的 4 个孔(1,3,5,6)分别接数据、GND、V_{CC}和时钟。还有两个没有使用的孔(2,8)保留为备用的数据和时钟端，在有些计算机或者 FPGA 上，只有一个 PS/2 接口，此时可以利用这两个没有使用的针/孔，用一个 Y 型的连接器同时连接鼠标和键盘。

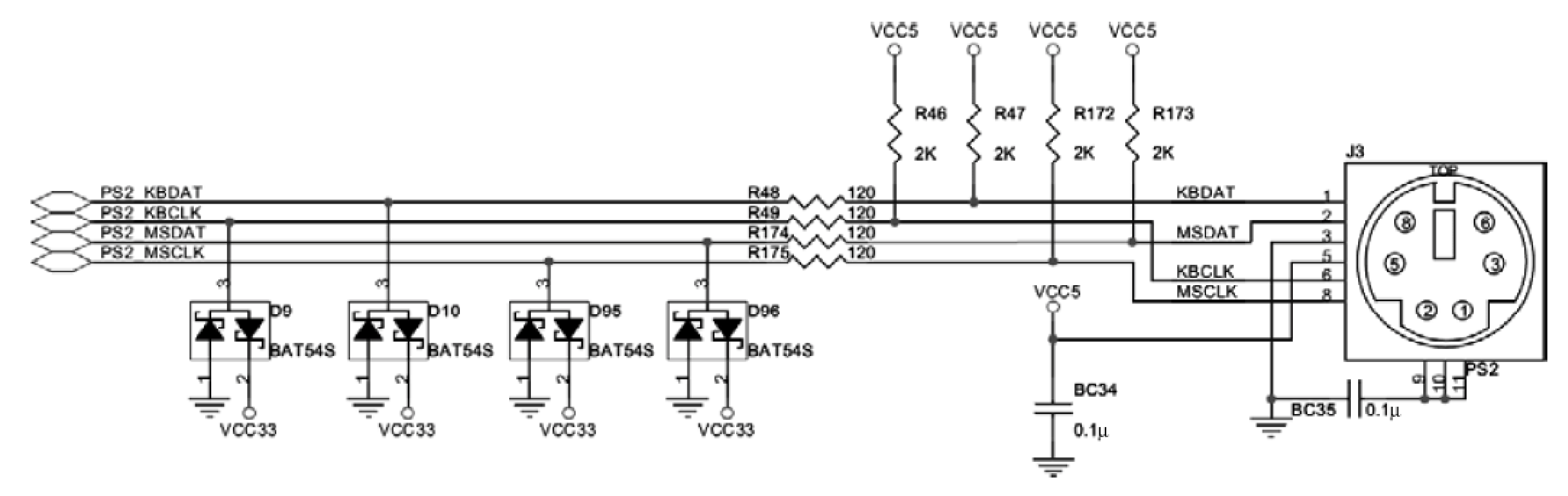


图 6-6 DE2-70 开发板上 PS/2 接口内部电路

PS/2 接口线和 FPGA 的外部引脚连接如表 6-2 所示。

表 6-2　PS/2 接口线和 FPGA 的外部引脚连接

Signal Name	FPGA Pin No.	Description
PS2_KBCLK	PIN_F24	PS/2 Clock
PS2_KBDAT	PIN_E24	PS/2 Data
PS2_MSCLK	PIN_D26	PS/2 Clock(reserved for second PS/2 device)
PS2_MSDAT	PIN_D25	PS/2 Data(reserved for second PS/2 device)

6.1.3　PS/2 键盘控制器的设计

以下为接收键盘数据的 Verilog HDL 代码，此代码只负责接收键盘送来的数据，如何识别出按下的到底是什么按键由软件来处理。如何显示出这些数据或者键符也请读者自行设计。

这里设置了一个 8 字节的缓冲区，以防止数据丢失。首先检测时钟的下降沿，然后开始逐位接收数据并放入缓冲区。缓冲区是一个先进先出的队列，配有写指针和读指针。当队列不空时，送出 ready 信号，表示此时有键按下；当队列溢出时，送出 overflow 信号。

程序清单 6.1　键盘控制器。

```
module ps2_keyboard(clk,clrn,ps2_clk,ps2_data,data,ready,
overflow,count);
   input clk,clrn,ps2_clk,ps2_data;
   output [7:0] data;
   reg [7:0] data;
   output ready;
   reg ready;
   output reg overflow;            //fifo overflow
   output reg [3:0] count;         //count ps2_data bits
      //internal signal, for test
   reg [9:0] buffer;               //ps2_data bits
   reg [7:0] fifo[7:0];            //data fifo
   reg [2:0] w_ptr,r_ptr;          //fifo write and read pointers
      //detect falling edge of ps2_clk
   reg [2:0] ps2_clk_sync;
   always @ (posedge clk)
      begin
      ps2_clk_sync<= {ps2_clk_sync[1:0],ps2_clk};
      end
```

```
    wire sampling=ps2_clk_sync[2] & ~ps2_clk_sync[1];
    always @ (posedge clk)
        begin
        if (clrn==0)                              //reset
            begin
            count<=0; w_ptr<=0; r_ptr<=0; overflow<=0;
            end
        else
            if (sampling)
                begin
                if (count==4'd10)
                    begin
                    if ((buffer[0]==0) &&         //start bit
                            (ps2_data)     &&     //stop bit
                            (^buffer[9:1]))       //odd  parity
                        begin
                        fifo[w_ptr]<=buffer[8:1];         //kbd scan code
                        w_ptr<=w_ptr+3'b1;
                        ready<=1'b1;
                        overflow<=overflow|(r_ptr==(w_ptr+3'b1));
                        end
                    count<=0;                             //for next
                    end
                    else
                        begin
                        buffer[count]<=ps2_data;          //store ps2_data
                        count<=count+3'b1;
                      end
                end
        if (ready)                                          //read to output next data
            begin
            data=fifo[r_ptr];
            r_ptr<=r_ptr+3'd1;
            ready<=1'b0;
            end
        end
endmodule
```

图 6-7 是这段代码的仿真波形，显示的是接收左“Shift”键和“W”键的扫描码 12H 和 1DH 的情形。请注意，以接收 12H 为例，PS/2 接口数据传送顺序为：起始位(1'b0)＋8 位数据位(由低到高)＋奇校验位＋停止位(1'b1)，那么传送 12H 时从 kb_data 端送出的数据顺序应该为 0010 0100 011。

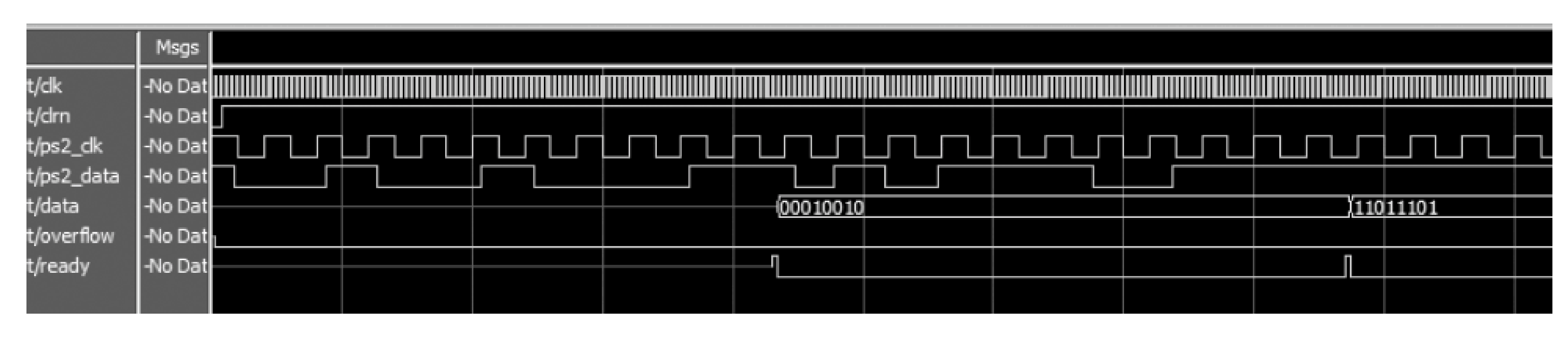


图 6-7　PS/2 键盘仿真波形

6.2　LCD 接口原理及实现

LCD 是基于液晶电光效应的显示器件，液晶显示器以其体积小、功耗低及使用灵活等特点而应用广泛。在我们的生活中随处可以见到液晶显示器，例如电脑、电子表、家用电器及 MP3 等。液晶显示器可以分为两大类，一类是点阵型的，另一类是字符型的。点阵型液晶显示器通常面积大，可以显示图形、图像等。而字符型液晶显示器是一种用点阵图形来显示字符的液晶显示器，它面积小，只能显示字符和一些很简单的图形，一般只有两行，简单易控制且成本低，在生活中也有广泛应用。

本实验的目的是学习字符型 LCD 工作原理，学习 LCD 接口控制器的设计方法。

6.2.1　LCD 简介

6.2.1.1　LCD 的外观和引脚功能

DE2-70 开发板上配置了一块 CFAH1602B-TMC-JP 型液晶字符型显示器件，它是 2 行 16 字的字符显示器件，可以显示 192 种 (5×7)字符、32 种(5×10)字符、自编 8 种 (5×7)或 4 种(5×10)字符，如图 6-8 所示。



图 6-8　CFAH1602B-TMC-JP 型液晶字符型显示器

该显示器的工作电压为 5V，芯片的引脚通常为 14 个引脚或者 16 个引脚。16 个引脚除了比 14 个引脚多两个电源引脚(V_{CC} 和 GND)外，其控制原理和 14 个引脚的器件一样。CFAH1602B-TMC-JP 型液晶显示器的引脚及其功能描述如表 6-3 所示。

CFAH1602B-TMC-JP 型液晶显示器内部有一个控制器 HD44780，产生控制信号，控制液晶显示器的显示。用户所有的指令都是发送给控制器 HD44780 的，显示器内部的工作由控制器协调。

表 6-3 CFAH1602B-TMC-JP 型液晶显示器的引脚及其功能

引 脚 号	信 号	功 能 描 述
1	V_{ss}	Ground
2	V_{dd}	Supply Voltage for logic
3	V_o	Operating voltage for LCD
4	RS	H:DATA, L:Instruction code
5	R/W	H:Read(MPU→Module)L:Write(MPU→Module)
6	E	Chip enable signal
7～14	DB_0～DB_7	Data bit 0～Data bit 7
15	A/V_{ee}	Power supply for LED backlight(＋)
16	K	Power supply for LED backlight(－)

6.2.1.2 LCD 的结构

图 6-9 是 CFAH1602B-TMC-JP 型液晶显示器的结构模块图，LCD 模块专门用于显示，控制器 HD44780 产生控制信号，控制 LCD 的显示，控制器 HD44780 受片外微处理器的控制，微处理器输出适当的控制信号给 LCD 片内控制器，控制 LCD 的显示，此微处理器由用户选择和设计。

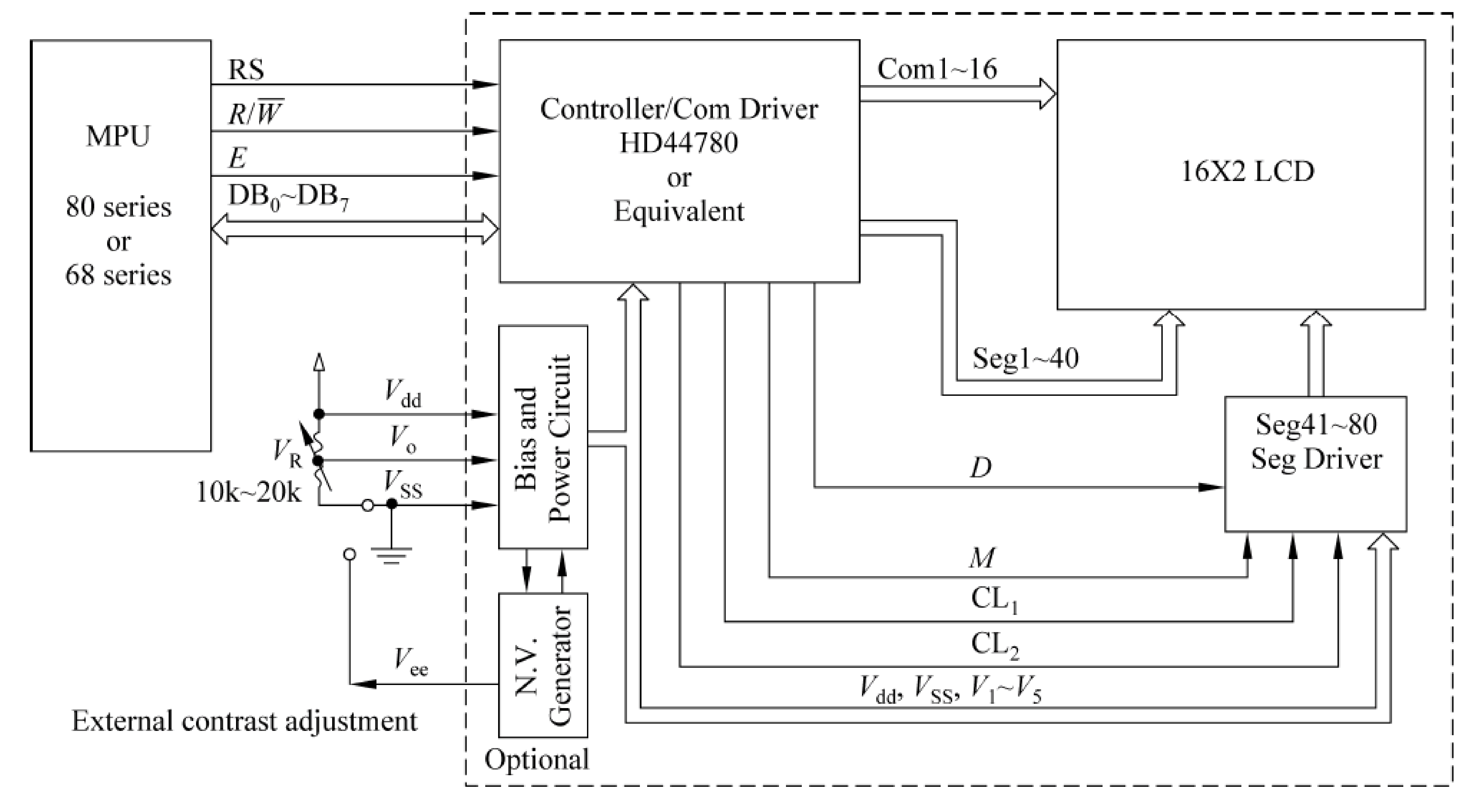


图 6-9 CFAH1602B-TMC-JP 型液晶显示器的结构模块图

6.2.2 LCD 与 FPGA 的连接

DE2-70 开发板上的液晶显示器的外部引脚和 FPGA 的输入输出引脚直接相连，用户可以利用 FPGA 来实现控制 LCD 的微处理器。LCD 在 DE2-70 开发板上的外围电路，以及它与 FPGA 的引脚连接情况如图 6-10 所示。

表 6-4 是 LCD 的引脚和 FPGA 的输入输出引脚的连接表。

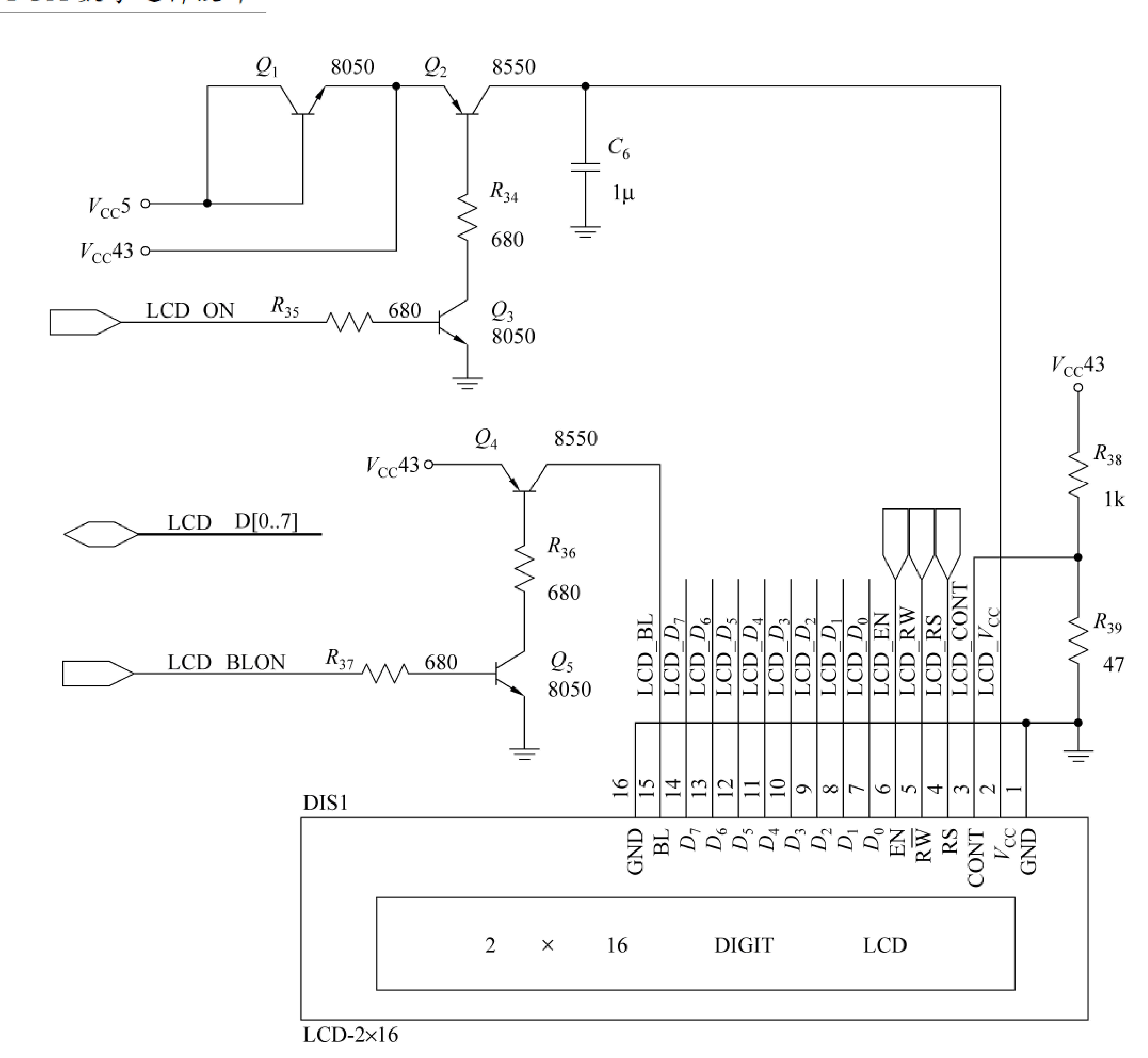


图 6-10　LCD 在 DE2-70 开发板上的外围电路

表 6-4　LCD 的引脚和 FPGA 的输入输出引脚的连接

LCD 引脚	FPGA 引脚	功 能 描 述
LCD_D[0]	PIN_E1	LCD_DATA[0]
LCD_D[1]	PIN_E3	LCD_DATA[1]
LCD_D[2]	PIN_D2	LCD_DATA[2]
LCD_D[3]	PIN_D3	LCD_DATA[3]
LCD_D[4]	PIN_C1	LCD_DATA[4]
LCD_D[5]	PIN_C2	LCD_DATA[5]
LCD_D[6]	PIN_C3	LCD_DATA[6]
LCD_D[7]	PIN_B2	LCD_DATA[7]
oLCD_RW	PIN_F3	LCD Read/Write Select，0=Write,1=Read
oLCD_EN	PIN_E2	LCD Enable
oLCD_RS	PIN_F2	LCD Command/Data Select，0=Command,1=Data
oLCD_ON	PIN_F1	LCD Power ON/OFF
oLCD_BLON	PIN_G3	LCD Back Light ON/OFF

6.2.3 LCD 的控制器 HD44780

6.2.3.1 HD44780 内的存储器和寄存器

HD44780 有内置存储器，用于存储可显示的字符，CGROM(Character Generator ROM)是字符产生器，存放标准 ASCII 码；另外还有一块用于存放用户自定义字符的存储空间，称为 CGRAM(Character Generator RAM)。它们在存储器内被统一编址，如表 6-5 所示。

表 6-5　HD44780 内置存储器编址

地　址	功　能
0x00～0x0F	自定义的字符图形 RAM(8 组 5×8 点阵的字符，或者 4 组 5×10 点阵的字符)
0x10～0x1F	未定义
0x20～0x7F	字符产生器 CGROM，存放标准的 ASCII 码
0x80～0x9F	未定义
0xA0～0xFF	字符产生器 CGROM，存放日文字符和希腊文字符

除了 CGROM 和 CGRAM 外，LCD 内部还有 DDRAM(Display Data RAM)，用于存放等待显示的内容(待显示的字符码，即 CGROM 中此字符的地址)。DDRAM 中的内容是等待传送到 LCD 中显示的字符地址，而 DDRAM 的地址是此字符要显示到 LCD 上的位置，16×2 的字符型 LCD 的 DDRAM 地址与显示位置的对应关系如表 6-6 所示。

表 6-6　DDRAM 地址与显示位置

00H	01H	02H	03H	04H	05H	06H	07H	08H	09H	0AH	0BH	0CH	0DH	0EH	0FH
40H	41H	42H	43H	44H	45H	46H	47H	48H	49H	4AH	4BH	4CH	4DH	4EH	4FH

LCD 控制器的指令系统规定：送待显示字符代码的指令之前，先要送 DDRAM 的地址。即如果希望在 LCD 的某一特定位置显示某一特定字符，要先指定该字符要显示的位置，再写入该字符在存储器中的存储地址。

LCD 内部控制器内有两个 8 位寄存器，一个指令寄存器和一个数据寄存器。

指令存储器(IR)存储指令代码，如清除显示、指针移位、寻址显示数据 RAM(DDRAM)的信息和字符产生器 RAM(CGRAM)的信息等。指令由微处理器(就是我们要设计的处理器)写入控制器中。数据寄存器(DR)暂存要写到 DDRAM 和 CGRAM 的数据，或者从中读出的数据。读写时，当地址写入 IR 时，数据就从 DR 存进 DDRAM(或 CGRAM)中了或从中读出了。通过寄存器选择位(RS)来选择两个寄存器(0-IR，1-DR)，请见表 6-7。

(1) 忙标志(BF)

当该标志位为 1 时，控制器工作于内部操作模式，不接收下一条指令。当 RS=0 而且 R/W=1 时，忙标志被输出到 DB_7，下一条指令只有在忙标志被清零时才能写入控制器。

表 6-7　RS 和 *R/W* 功能描述

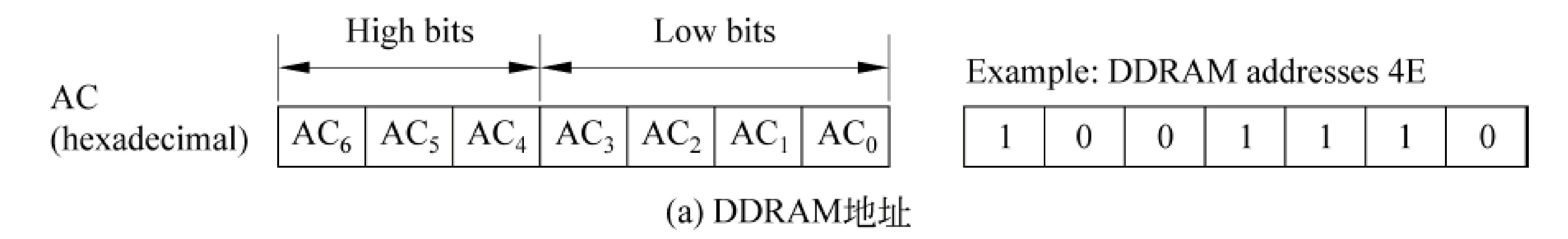
RS	*R/W*	功 能 描 述
0	0	IR write as an internal operation (display clear, etc.)
0	1	Read busy flag (DB_7) and address counter (DB_0 to DB_7)
1	0	Write data to DDRAM or CGRAM (DR to DDRAM or CGRAM)
1	1	Read data from DDRAM or CGRAM (DDRAM or CGRAM to DR)

(2) 地址计数器(AC)

地址计数器用于寄存 DDRAM 和 CGRAM 的地址。

(3) 显示数据 RAM(DDRAM)

DDRAM 用于存储 8 位的特征代码,它可以扩展为 80×8 位显示数据或 80 个字符,图 6-11 就是 DDRAM 地址和液晶显示器显示位置的对应关系。



(a) DDRAM地址

Display position DDRAM address

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

(b) 2行×16字符显示

图 6-11　DDRAM 地址和液晶显示器显示位置的对应关系

(4) 字符产生器 ROM(CGROM)

CGROM 从 8 位字符码中产生 5×8 个点或 5×10 个点的显示模式,如表 6-8 和表 6-10 所示。

(5) 字符产生器 RAM(CGRAM)

在 CGRAM 中,用户可以经过程序重写字符,能写 8 个 5×8 个点的字符模式,或者 4 个 5×10 个点的字符模式,请见表 6-9 和表 6-10。

表 6-9 和表 6-10 中显示的是存储在 CGRAM 中的要显示的字符码模式,其地址存储在 DDRAM 中,如表 6-9 的左栏中显示的数据。这张表显示了 CGRAM 地址、字符码(DDRAM)和字符模式(CGRAM 数据)之间的关系。

6.2.3.2　字符型液晶显示模块指令集

1. 清屏

RS	*R/W*	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
0	0	0	0	0	0	0	0	0	1

运行时间:(250kHz)1.64ms。

表 6-8　CGROM 8 位字符码表

Upper 4 bit / Lower 4 bit	LLLL	LLLH	LLHL	LLHH	LHLL	LHLH	LHHL	LHHH	HLLL	HLLH	HLHL	HLHH	HHLL	HHLH	HHHL	HHHH
LLLL	CG RAM (1)			0	@	P	`	p				ー	タ	ミ	α	p
LLLH	(2)		!	1	A	Q	a	q			。	ア	チ	ム	ä	q
LLHL	(3)		"	2	B	R	b	r			「	イ	ツ	メ	β	θ
LLHH	(4)		#	3	C	S	c	s			」	ウ	テ	モ	ε	∞
LHLL	(5)		$	4	D	T	d	t			、	エ	ト	ヤ	μ	Ω
LHLH	(6)		%	5	E	U	e	u			・	オ	ナ	ユ	σ	ü
LHHL	(7)		&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
LHHH	(8)		'	7	G	W	g	w			ァ	キ	ヌ	ラ	g	π
HLLL	(1)		(	8	H	X	h	x			ィ	ク	ネ	リ	√	x̄
HLLH	(2)		)	9	I	Y	i	y			ゥ	ケ	ノ	ル	⁻¹	y
HLHL	(3)		*	:	J	Z	j	z			ェ	コ	ハ	レ	j	千
HLHH	(4)		+	;	K	[	k	{			ォ	サ	ヒ	ロ	ˣ	万
HHLL	(5)		,	<	L	¥	l					ャ	シ	フ	ワ	¢	円
HHLH	(6)		-	=	M	]	m	}			ュ	ス	ヘ	ン	£	÷
HHHL	(7)		.	>	N	^	n	→			ョ	セ	ホ	゛	ñ	
HHHH	(8)		/	?	O	_	o	←			ッ	ソ	マ	゜	ö	█

表 6-9　5×8 个点的字符模式

Character Codes (DDRAM data)	CGRAM Address		Character Patterns (CGRAM data)					
7 6 5 4 3 2 1 0	5 4 3	2 1 0	7 6 5	4	3	2	1	0
High　Low	High	Low	High	Low				
		0 0 0	* * *					0
		0 0 1	* * *		0	0	0	
		0 1 0	* * *		0	0	0	
		0 1 1	* * *					0
0 0 0 0 * 0 0 0	0 0 0	1 0 0	* * *		0		0	0
		1 0 1	* * *		0	0		0
		1 1 0	* * *		0	0	0	
		1 1 1	* * *	0	0	0	0	0
		0 0 0	* * *		0	0	0	
		0 0 1	* * *	0		0		0
		0 1 0	* * *					
		0 1 1	* * *	0	0		0	0
0 0 0 0 * 0 0 1	0 0 1	1 0 0	* * *					
		1 0 1	* * *	0	0		0	0
		1 1 0	* * *	0	0		0	0
		1 1 1	* * *	0	0	0	0	0
		0 0 0	* * *					
		0 0 1						
0 0 0 0 * 1 1 1	1 1 1	1 0 0						
		1 0 1						
		1 1 0						
		1 1 1	* * *					

表 6-10　5×10 个点的字符模式

Character Codes (DDRAM data)	CGRAM Address		Character Patterns (CGRAM data)					
7 6 5 4 3 2 1 0	5 4	3 2 1 0	7 6 5	4	3	2	1	0
High　Low	High	Low	High	Low				
		0 0 0 0	* * *	0	0	0	0	0
		0 0 0 1	* * *	0	0	0	0	0
		0 0 1 0	* * *		0			0
		0 0 1 1	* * *			0	0	
		0 1 0 0	* * *		0	0	0	
0 0 0 0 * 0 0 0	0 0	0 1 0 1	* * *		0	0	0	
		0 1 1 0	* * *					0
		0 1 1 1	* * *		0	0	0	0
		1 0 0 0	* * *		0	0	0	0
		1 0 0 1	* * *		0	0	0	0
		1 0 1 0	* * *	0	0	0	0	0
		1 1 1 1	* * *	*	*	*	*	*

功能：(1) 清除液晶显示器，即将 DDRAM 的内容全部填入“空白”的 ASCII 码 20H；
(2) 光标归位，即将光标撤回液晶显示屏的左上方；
(3) 将地址计数器(AC)的值设为 0。

2. 光标归位

RS	*R/W*	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
0	0	0	0	0	0	0	0	1	*

运行时间：(250kHz)1.64ms。
功能：(1) 把光标撤回到显示器的左上方；
(2) 把地址计数器(AC)的值设置为 0；
(3) 保持 DDRAM 的内容不变。

3. 输入方式设置

RS	*R/W*	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
0	0	0	0	0	0	0	1	I/D	S

运行时间：(250kHz)40μs。
功能：设定每次写入 1 个数据后光标的移位方向，并且设定每次写入一个字符后是否移动。
其中：$I/D=1$ 数据读写操作后 AC 自动增一；
$I/D=0$ 数据读写操作后 AC 自动减一；
$S=1$ 写入新数据后显示屏整体右移 1 个字符；
$S=0$ 写入新数据后显示屏不移动。

4. 显示开关控制

RS	*R/W*	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
0	0	0	0	0	0	1	D	C	B

运行时间：(250kHz)40μs。
功能：控制显示器开/关、光标显示/关闭以及光标是否闪烁。
其中：D 表示显示开关，$D=1$ 为开，$D=0$ 为关；
C 表示光标开关，$C=1$ 为开，$C=0$ 为关；
B 表示闪烁开关，$B=1$ 为开，$B=0$ 为关。

5. 光标、画面移位

RS	*R/W*	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
0	0	0	0	0	1	S/C	R/L	*	*

运行时间：(250kHz)40μs。

功能：使光标移位或使整个显示屏幕移位，光标画面移动不影响 DDRAM。

其中：$S/C=1$ 显示器上字符全部(画面)平移一个字符位，且 AC 值加 1；

$S/C=0$ 光标平移一个字符位，且 AC 值加/减 1；

$R/L=1$ 右移，$R/L=0$ 左移。

6. 功能设置

RS	R/W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
0	0	0	0	1	DL	N	F	*	*

运行时间：(250kHz)40μs。

功能：工作方式设置初始化指令，设定数据总线位数、显示的行数及字形。

其中：DL=1，8 位数据接口；DL=0，4 位数据接口。

$N=1$，两行显示；$N=0$，一行显示。

$F=1$，5×10 点阵字符/每字符；$F=0$，5×7 点阵字符/每字符。

7. CGRAM 地址设置

RS	R/W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
0	0	0	1	A_5	A_4	A_3	A_2	A_1	A_0

运行时间：(250kHz)40μs。

功能：设置下一个要存入数据的 CGRAM 地址，$A_5 \sim A_0 = 0 \sim 3$FH。

8. DDRAM 地址设置

RS	R/W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
0	0	1	A_6	A_5	A_4	A_3	A_2	A_1	A_0

运行时间：(250kHz)40μs。

功能：设置设定下一个要存入数据的 DDRAM 的地址。

9. 读 BF 及 AC 值

RS	R/W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
0	1	BF	AC_6	AC_5	AC_4	AC_3	AC_2	AC_1	AC_0

运行时间：(250kHz)40μs。

功能：读忙 BF 值和地址计数器 AC 值。

其中：(1) 读取忙碌信号 BF 的内容。BF=1 表示液晶显示器忙，暂时无法接收控制器送来的数据或指令；当 BF=0 时，液晶显示器可以接收控制器送来的数据或指令。

(2) 读取地址计数器(AC)内容，此时 AC 值意义为最近一次地址设置(CGRAM 或 DDRAM)定义。

10. 写数据,数据写入 DDRAM 或 CGRAM

RS	*R/W*	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
1	0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

运行时间:(250kHz)40μs。

功能:(1) 将字符码写入 DDRAM,以使液晶显示屏显示出相对应的字符。

(2) 将使用者自己设计的图形存入 CGRAM。

11. 读数据,从 CGRAM 或 DDRAM 读出数据

RS	*R/W*	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0
1	1	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

运行时间:(250kHz)40μs。

功能:根据最近设置的地址性质从 DDRAM 或 CGRAM 将数据读出。

细心的读者肯定发现了,在上面的指令集中,有 RS、*R/W* 和 8 位数据总线,却少了一个使能位 *E*。使能位 *E* 对执行 LCD 指令起着关键作用,*E* 有两个有效状态,高电平(1)和下降沿(1→0)。当 *E* 为高电平时,如果 *R/W* 为 0,则 LCD 从微控制器读入指令或者数据;如果 *R/W* 为 1,则微控制器可以从 LCD 中读出状态字(BF 忙状态)和地址。而 *E* 的下降沿指示 LCD 执行其读入的指令或者显示其读入的数据。

图 6-12 是 HD44780 的写操作时序。

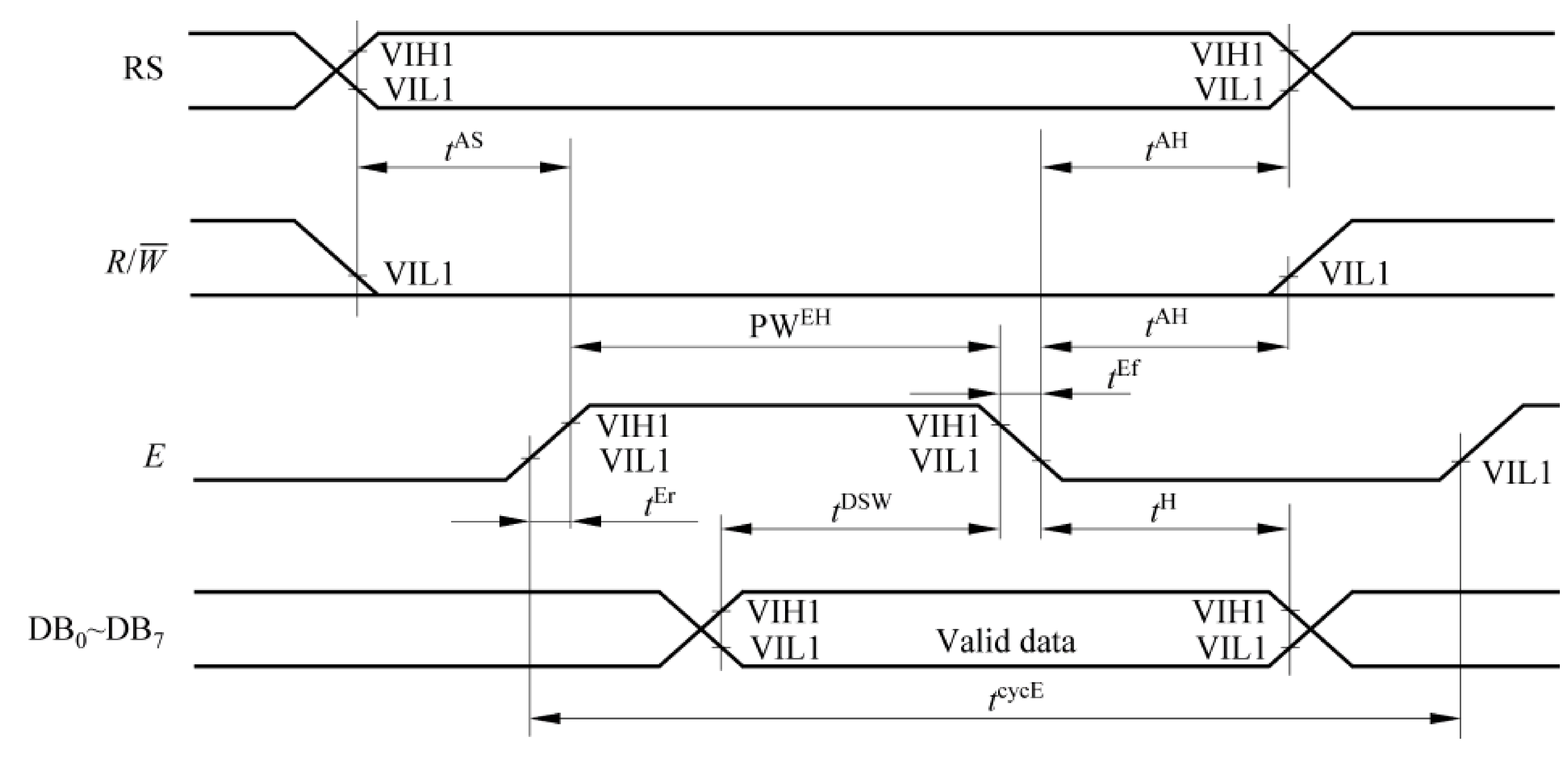


图 6-12 LCD 写操作时序

图 6-13 是 HD44780 的读操作时序。

6.2.4 LCD 显示控制器的设计

以下为 LCD 显示控制器的 Verilog HDL 代码,此段代码实现了在 LCD 上显示一串字符的方法。只要对代码稍加改动,就可以在 LCD 上显示自己想要显示的字符。

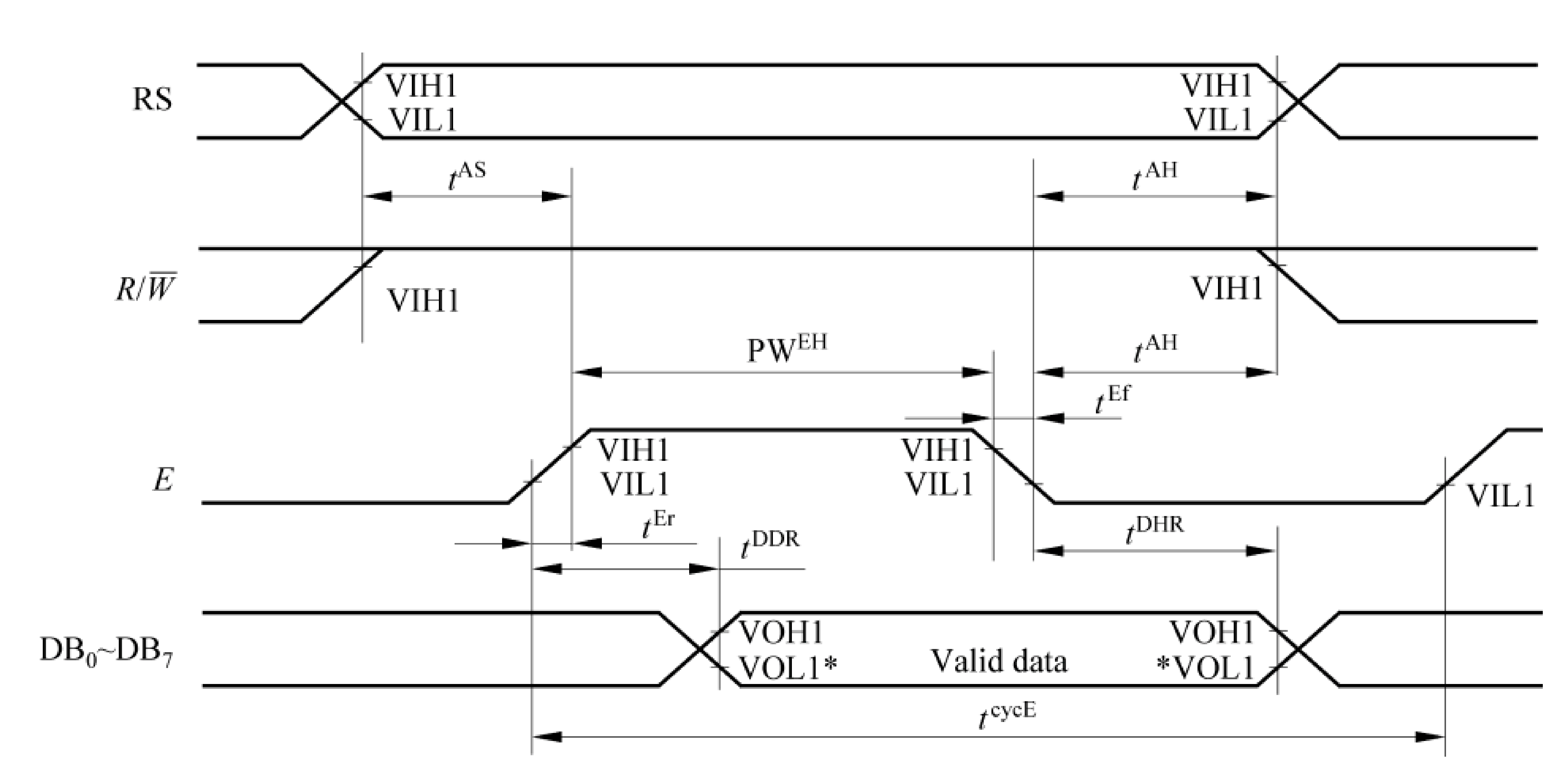


图 6-13　LCD 读操作时序

程序清单 6.2　LCD 控制器代码。

```
module LCD(
input   iCLK_50,
input   [0:0] iSW,
output  [7:0]LCD_D,
output  oLCD_RW,
output  oLCD_EN,
output  oLCD_RS,
output  oLCD_ON,
output  oLCD_BLON
);
//  Internal Wires/Registers
reg[5:0]   LUT_INDEX;
reg[8:0]   LUT_DATA;
reg[5:0]   mLCD_ST;
reg[17:0]  mDLY;
reg        mLCD_Start;
reg[7:0]   mLCD_D;
reg        moLCD_RS;
wire       mLCD_Done;

parameter   LCD_INTIAL= 0;
parameter   LCD_LINE1= 5;
parameter   LCD_CH_LINE= LCD_LINE1+ 16;
```

```
parameter   LCD_LINE2=LCD_LINE1+16+1;
parameter   LUT_SIZE=LCD_LINE1+32+1;                //39
assign oLCD_ON=1'b1;
assign oLCD_BLON=1'b1;
always@ (posedge iCLK_50)
begin
    if(!iSW[0])
    begin
        LUT_INDEX   <=0;
        mLCD_ST     <=0;
        mDLY        <=0;
        mLCD_Start  <=0;
        mLCD_D      <=0;
        moLCD_RS    <=0;
    end
    else
    begin
        if(LUT_INDEX<LUT_SIZE)
        begin
            case(mLCD_ST)
            0: begin
                    mLCD_D      <=LUT_DATA[7:0];
                    moLCD_RS    <=LUT_DATA[8];
                    mLCD_Start  <=1;
                    mLCD_ST     <=1;
                end
            1: begin
                    if(mLCD_Done)
                    begin
                        mLCD_Start  <=0;
                        mLCD_ST     <=2;
                    end
                end
            2: begin
                    if(mDLY<18'h3FFFE)
                    mDLY<=mDLY+1;
                    else
                    begin
```

```
                        mDLY      <= 0;
                        mLCD_ST   <= 3;
                    end
                end
            3: begin
                    LUT_INDEX     <= LUT_INDEX + 1;
                    mLCD_ST       <= 0;
                end
            endcase
        end
    end
end
//always
always    begin
    case (LUT_INDEX)
    //Initial
    LCD_INTIAL + 0: LUT_DATA  <= 9'h038;     //工作方式设置
    LCD_INTIAL + 1: LUT_DATA  <= 9'h00C;     //控制显示器开/关设置
    LCD_INTIAL + 2: LUT_DATA  <= 9'h001;     //清屏
    LCD_INTIAL + 3: LUT_DATA  <= 9'h006;     //输入方式设置
    LCD_INTIAL + 4: LUT_DATA  <= 9'h080;     //设定下一数据的 DDRAM 地址
    //Line 1
    LCD_LINE1 + 0: LUT_DATA   <= 9'h148;     //H
    LCD_LINE1 + 1: LUT_DATA   <= 9'h169;     //i
    LCD_LINE1 + 2: LUT_DATA   <= 9'h121;     //!
    LCD_LINE1 + 3: LUT_DATA   <= 9'h120;
    LCD_LINE1 + 4: LUT_DATA   <= 9'h157;     //W
    LCD_LINE1 + 5: LUT_DATA   <= 9'h165;     //e
    LCD_LINE1 + 6: LUT_DATA   <= 9'h16C;     //l
    LCD_LINE1 + 7: LUT_DATA   <= 9'h163;     //c
    LCD_LINE1 + 8: LUT_DATA   <= 9'h16F;     //o
    LCD_LINE1 + 9: LUT_DATA   <= 9'h16D;     //m
    LCD_LINE1 + 10: LUT_DATA  <= 9'h165;     //e
    LCD_LINE1 + 11: LUT_DATA  <= 9'h120;
    LCD_LINE1 + 12: LUT_DATA  <= 9'h174;     //t
    LCD_LINE1 + 13: LUT_DATA  <= 9'h16F;     //o
    LCD_LINE1 + 14: LUT_DATA  <= 9'h120;
    LCD_LINE1 + 15: LUT_DATA  <= 9'h120;
```

```
        //Change Line
        LCD_CH_LINE: LUT_DATA    <= 9'h0C0;     //设定下一数据的 DDRAM 地址
        //Line 2
        LCD_LINE2+ 0: LUT_DATA   <= 9'h120;     //
        LCD_LINE2+ 1: LUT_DATA   <= 9'h120;     //
        LCD_LINE2+ 2: LUT_DATA   <= 9'h120;     //
        LCD_LINE2+ 3: LUT_DATA   <= 9'h120;     //
        LCD_LINE2+ 4: LUT_DATA   <= 9'h14E;     //N
        LCD_LINE2+ 5: LUT_DATA   <= 9'h14A;     //J
        LCD_LINE2+ 6: LUT_DATA   <= 9'h155;     //U
        LCD_LINE2+ 7: LUT_DATA   <= 9'h120;     //
        LCD_LINE2+ 8: LUT_DATA   <= 9'h143;     //C
        LCD_LINE2+ 9: LUT_DATA   <= 9'h153;     //S
        LCD_LINE2+ 10: LUT_DATA  <= 9'h120;
        LCD_LINE2+ 11: LUT_DATA  <= 9'h120;     //
        LCD_LINE2+ 12: LUT_DATA  <= 9'h144;     //D
        LCD_LINE2+ 13: LUT_DATA  <= 9'h145;     //E
        LCD_LINE2+ 14: LUT_DATA  <= 9'h150;     //P
        LCD_LINE2+ 15: LUT_DATA  <= 9'h121;     //!
        default: LUT_DATA        <= 9'h000;
        endcase
    end
    LCD_Controller   u0(   //Host Side
                           .iDATA(mLCD_D),
                           .iRS(moLCD_RS),
                           .iStart(mLCD_Start),
                           .oDone(mLCD_Done),
                           .iCLK(iCLK_50),
                           .iRST(iSW[0]),
                           //LCD Interface
                           .LCD_D(LCD_D),
                           .oLCD_RW(oLCD_RW),
                           .oLCD_EN(oLCD_EN),
                           .oLCD_RS(oLCD_RS));
    endmodule
    //To display one letter on LCD
    module LCD_Controller (   //Host Side
                              iDATA,iRS,
```

```
                        iStart,oDone,
                        iCLK,iRST,
                        //LCD Interface
                        LCD_D,
                        oLCD_RW,
                        oLCD_EN,
                        oLCD_RS);
//CLK
parameter CLK_Divide=16;
//Host Side
input   [7:0]   iDATA;
input   iRS,iStart;
input   iCLK,iRST;
output  reg  oDone;
//LCD Interface
output  [7:0]  LCD_D;
output  reg    oLCD_EN;
output         oLCD_RW;
output         oLCD_RS;
//Internal Register
reg   [4:0]   Cont;
reg   [1:0]   ST;
reg   preStart,mStart;
/////////////////////////////////////////////
//Only write to LCD, bypass iRS to oLCD_RS
assign  LCD_D     =iDATA;
assign  oLCD_RW   =1'b0;
assign  oLCD_RS   =iRS;
/////////////////////////////////////////////
always@ (posedge iCLK)
begin
    if(!iRST)
    begin
        oDone     <=1'b0;
        oLCD_EN   <=1'b0;
        preStart  <=1'b0;
        mStart    <=1'b0;
        Cont      <=0;
```

```
        ST          <=0;
    end
    else
    begin
        //////Input Start Detect///////
        preStart<=iStart;
        if({preStart,iStart}==2'b01)
        begin
            mStart   <=1'b1;
            oDone    <=1'b0;
        end
        //////////////////////////////////
        if(mStart)
        begin
            case(ST)
            0: ST   <=1;                    //Wait Setup
            1: begin
                    oLCD_EN   <=1'b1;
                    ST        <=2;
                end
            2: begin
                    if(Cont<CLK_Divide)
                    Cont      <=Cont+1;
                    else
                    ST        <=3;
                end
            3: begin
                    oLCD_EN   <=1'b0;
                    mStart    <=1'b0;
                    oDone     <=1'b1;
                    Cont      <=0;
                    ST        <=0;
                end
            endcase
        end
    end
end
endmodule
```

6.3 VGA 接口原理及实现

VGA 接口是 IMB 制定的一种视频数据的传输标准，是电脑显示器最典型的接口。本实验的目的是学习 VGA 接口原理，学习 VGA 接口控制器的设计方法。

6.3.1 VGA 简介

6.3.1.1 VGA 接口的外观和引脚功能

VGA(Video Graphics Array)接口，即视频图形阵列。VGA 接口最初是用于连接 CRT 显示器的接口，CRT 显示器因为设计制造上的原因，只能接受模拟信号输入，这就需要显卡能输出模拟信号，VGA 接口就是显卡上输出模拟信号的接口，在传统的 CRT 显示器中，使用的都是 VGA 接口。VGA 接口是 15 针/孔的梯形插头，分成 3 排，每排 5 个，如图 6-14 所示。

VGA 接口的接口信号主要有 5 个：R(Red)、G(Green)、B(Blue)、HS(Horizontal Synchronization)和 VS(Vertical Synchronization)，即红、绿、蓝、垂直同步和水平同步(也称行同步和帧同步)。

6.3.1.2 VGA 的工作原理

图像的显示是以像素(点)为单位，显示器的分辨率是指屏幕每行有多少个像素及每帧有多少行，标准的 VGA 分辨率是 640×480，也有更高的分辨率，如 1024×768、1280×1024、1920×1200 等。从人眼的视觉效果考虑，屏幕刷新的频率(每秒钟显示的帧数)应该大于 24，这样屏幕看起来才不会闪烁，VGA 显示器一般的刷新频率是 60Hz。

每一帧图像的显示都是从屏幕的左上角开始一行一行进行的，行同步信号是一个负脉冲，行同步信号有效后，由 RGB 端送出当前行显示的各像素点的 RGB 电压值，当一帧显示结束后，由帧同步信号送出一个负脉冲，重新开始从屏幕的左上端开始显示下一帧图像，如图 6-15 所示。

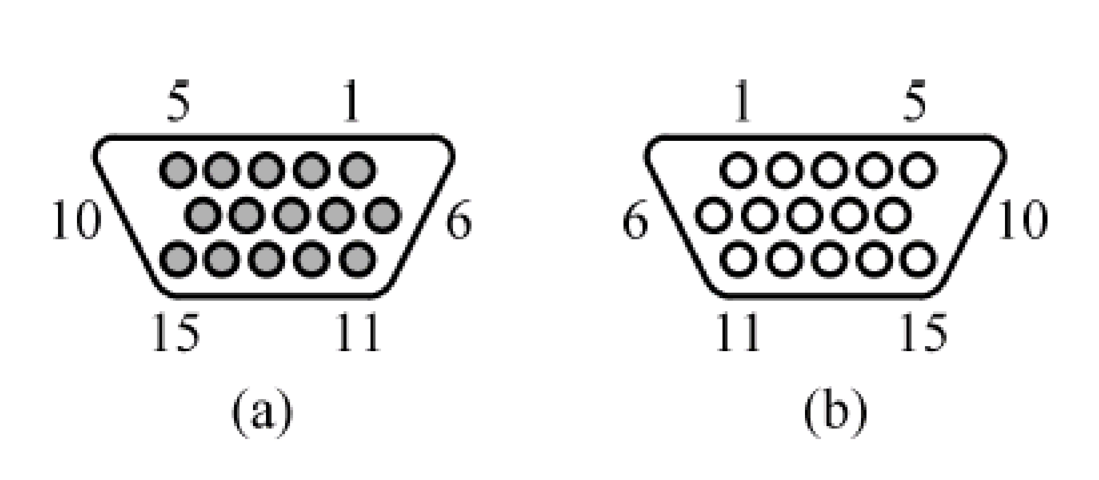


图 6-14 VGA 接口形状示意图

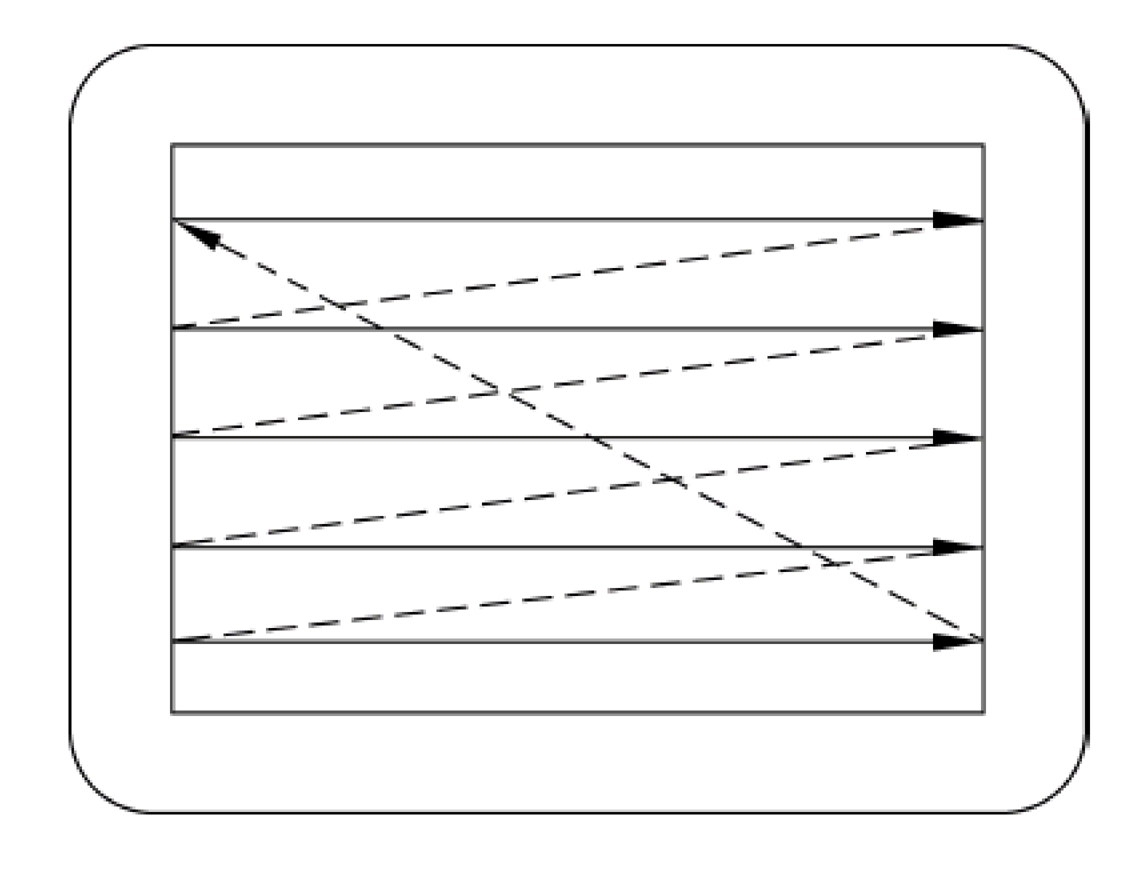


图 6-15 显示器扫描示意图

RGB 端并不是所有时间都在传送像素信息，由于 CRT 的电子束从上一行的行尾到下一行的行头需要时间，从屏幕的右下角回到左上角开始下一帧也需要时间，这时 RGB

送的电压值为0(黑色),这些时间称为电子束的行消隐时间和场消隐时间,行消隐时间以像素为单位,帧消隐时间以行为单位。VGA行扫描、场扫描时序示意图如图6-16所示。



图6-16　VGA行扫描、场扫描时序示意图

由图6-16可看出,有效地显示一行信号需要96+48+640+16=800个像素点的时间,其中行同步负脉冲宽度为96个像素点时间,行消隐后沿需要48个像素点时间,每行显示640个像素点,行消隐前沿需要16个像素点的时间,一行显示时间为640个像素点时间,一行消隐时间为160个像素点时间。

有效显示一帧图像需要2+33+480+10=525行时间,其中场同步负脉冲宽度为2个行显示时间,场消隐后沿需要33个行显示时间,每场显示480行,场消隐前沿需要10个行显示时间,一帧显示时间为480行显示时间,一帧消隐时间为45行显示时间。

6.3.2　VGA和FPGA的连接

根据VGA的工作原理,VGA工作时的同步脉冲HS和VS是一个负脉冲,是由FPGA产生的。但RGB是电压值,是模拟信号,如何产生呢?

数模转换器ADV7123是一款单芯片、三通道、高速数模转换器。其内置三个高速、10位、带互补输出的视频数模转换器、一个标准TTL输入接口以及一个高阻抗、模拟输出电流源。DE2-70开发平台用它产生显示器的模拟信号(红、绿、蓝),ADV7123连接在Cyclone Ⅱ FPGA和VGA接口之间,用于将数字RGB信号转换成模拟电压输出。其电路连接图如图6-17所示。

ADV7123上的工作时钟、控制信号和数据来自于Cyclone Ⅱ FPGA,$R_0 \sim R_9$、$G_0 \sim G_9$和$B_0 \sim B_9$是三组来自于FPGA的数据,用于控制RGB的输出电压,显示不同颜色。VGA_BLANK_N控制信号低电平有效,用于控制ADV7123的模拟信号输出,当VGA_BLANK_N为低电平时,$R_0 \sim R_9$、$G_0 \sim G_9$和$B_0 \sim B_9$的输入像素信息无效。VGA_SYNC_N控制信号也是低电平有效,是ADV7123内部独立的视频同步控制输入端,一般设置为逻辑地。VGA_CLOCK是ADV7123的工作时钟,其频率取决于显示器的刷新频率(每秒显示刷新的帧数)和分辨率(每帧的像素点数)。根据要求,每秒显示60帧,每帧800×525个像素点,60×800×525=25200000,即每秒要显示25.2M个点,此时ADV7123的工作频率就为25.2MHz。

图 6-17　FPGA 和 VGA 接口电路图

6.3.3 VGA 显示控制器的设计

下面的代码是一个简单的用于图像显示的 VGA 的时序控制器，此段代码使屏幕显示为全屏绿色。为简单起见，代码中的频率采用 25MHz。

程序清单 6.3 VGA 显示控制器代码。

```
module  io_vga(
                iCLK_50,         //DE2_70  clock   50MHz
                iCLRN,           //clear_N  connect to iSW[0]
                 //to  ADV7123
                 oVGA_R,
                 oVGA_G,
                 oVGA_B,
                 oVGA_SYNC_N,
                 oVGA_BLANK_N,
                 oVGA_CLOCK,
                     //to  VGA
                 oVGA_HS,
                 oVGA_VS,);
input iCLK_50;
input iCLRN;
output [9:0] oVGA_R, oVGA_G, oVGA_B;
output oVGA_SYNC_N, oVGA_BLANK_N, oVGA_CLOCK;
output oVGA_HS, oVGA_VS;
reg  vga_clk;
reg [9:0] h_count,  v_count;
reg [11:0] data_reg;
wire video_out;
/* -------------------------------------------------------
        常量定义
------------------------------------------------------- */
    //Horizontal Parameter   (Pixel)
    parameter   H_SYNC_CYC    =96;
    parameter   H_SYNC_BACK   =48;
    parameter   H_SYNC_ACT    =640;
    parameter   H_SYNC_FRONT  =16;
    parameter   H_SYNC_TOTAL  =800;            //96+48+640+16=800
    //Virtical Parameter   (Line)
    parameter   V_SYNC_CYC    =2;
```

```
        parameter   V_SYNC_BACK  =32;
        parameter   V_SYNC_ACT   =480;
        parameter   V_SYNC_FRONT =11;
        parameter   V_SYNC_TOTAL =525;              //2+32+480+11=525
        //Start Offset
        parameter   X_START      =H_SYNC_CYC+H_SYNC_BACK;
//96+48=144   before   640
        parameter   Y_START      =V_SYNC_CYC+V_SYNC_BACK;
//2+32=34   before   480

        //color of back ground
        parameter Color_R=0;
        parameter Color_G=10'b1111111111;
        parameter Color_B=0;
/*---------------------------------------------*/
//oVGA_CLK   Generator,   50MHz to 25MHz
always @ (posedge iCLK_50 or negedge iCLRN) begin
    if(iCLRN==0)    vga_clk<=1'b1;
    else             vga_clk<=~vga_clk;
    end
//H_Sync Counter
always @ (posedge vga_clk or negedge iCLRN) begin
    if(iCLRN==0)                        h_count<=10'd0;
    else if (h_count==H_SYNC_TOTAL)     h_count<=10'd0;
        else                            h_count=h_count+10'd1;
    end
//V_Sync Counter
always @ (posedge vga_clk or negedge iCLRN) begin
    if(iCLRN==0)                            v_count<=10'd0;
    else if (h_count==H_SYNC_TOTAL) begin
            if(v_count==V_SYNC_TOTAL)       v_count<=10'd0;
            else                            v_count=v_count+10'd1;
            end
    end
assign    video_out = ((h_count>=X_START)
&&(h_count<X_START+H_SYNC_ACT))
&&((v_count>=Y_START)
&&(v_count<Y_START+V_SYNC_ACT));
```

```
assign oVGA_HS= (h_count>=10'd96);
assign oVGA_VS= (v_count>=10'd2);
assign oVGA_R= (video_out)?Color_R:0;
assign oVGA_G= (video_out)?Color_G:0;
assign oVGA_B= (video_out)?Color_B:0;
assign oVGA_SYNC_N=1'b0;
assign oVGA_BLANK_N=oVGA_HS & oVGA_VS;
assign oVGA_CLOCK=vga_clk;
endmodule
```

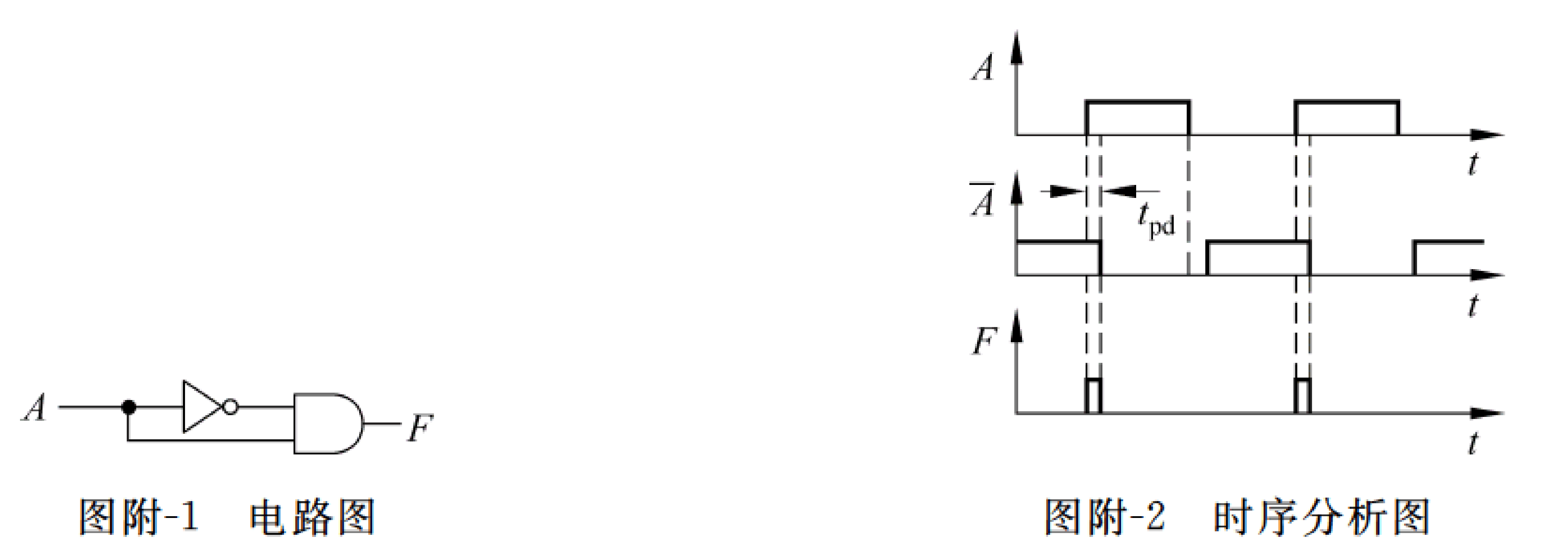
附录

竞争、冒险和毛刺

附.1　竞争、冒险和毛刺现象

在数字逻辑电路中，当一个门的输入端有两个或者两个以上的输入信号发生变化时，由于这些输入信号是经过不同的路径传输过来的，使得它们状态改变的时刻有先有后，这种由同一个门的输入信号改变时差所引起的现象称为竞争(Race)。竞争的结果如果导致了冒险(Hazard)发生(例如毛刺)，并造成错误的后果，那么这种竞争就称为临界竞争；如果竞争的结果没有导致冒险发生，或者虽然有冒险发生，但是不影响系统的性能，那么这种竞争就是非临界竞争。

在图附-1 中，逻辑上 $F=A\overline{A}$，F 的输出值应该恒为 0。仔细观察电路，电路中的与门有两个输入端 A 和 $\overline{A}$，当 A 发生改变时，A 的值被立刻送进了与门，而与门的另外一个输入端 $\overline{A}$ 则要经过一个非门的延时 $\overline{A}$ 值才被送进与门，当 A 由 0 变为 1 时，与门的 A 输入端立刻变为了 1，而 $\overline{A}$ 这个输入端要保持短暂的 1 值后才变为 0，这时，与门的输出将会出现一个短暂的 1 然后才输出常态值 0，如图附-2 所示，其中 t_{pd} 为非门的延时时间。



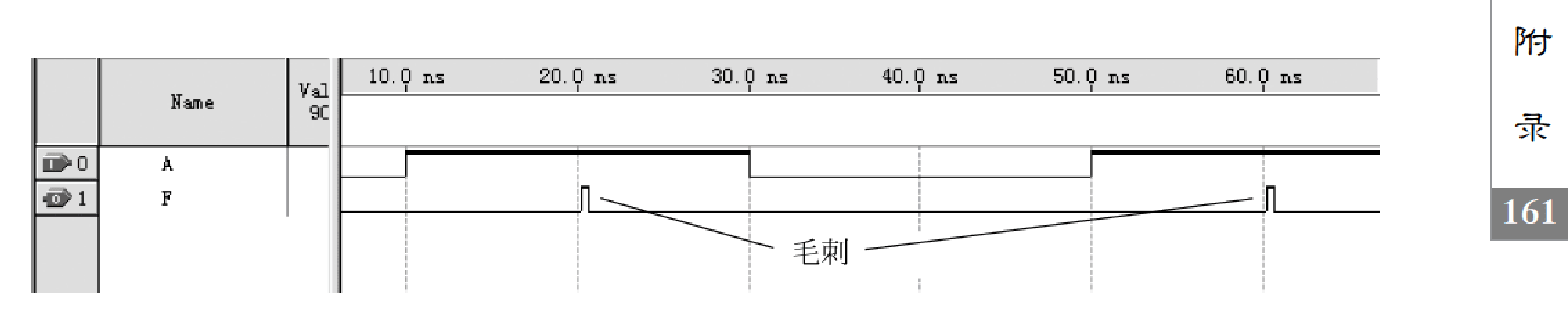
图附-1　电路图

图附-2　时序分析图

在 Quartus Ⅱ 中对电路 $F=A\overline{A}$ 进行时序仿真，也可以看见在 A 由 0 向 1 变化的瞬间，输出端 F 会产生毛刺，如图附-3 所示。

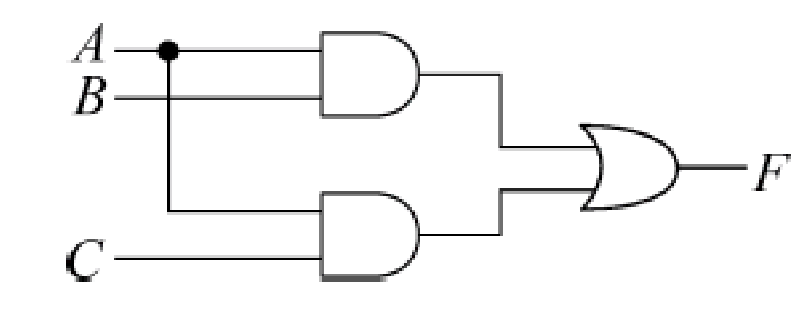
逻辑函数在单个参数发生改变时产生的冒险现象称为逻辑冒险。

当电路中同时有多个变量发生变化时，由于每一个变量的路径不同，到达同一个门的时间会有所差别，从而产生冒险。这一现象在采用 FPGA 设计数字逻辑电路中尤为明显。FPGA 芯片是由输入输出单元(IOE)、逻辑阵列块(LAB)和可编程连线(PIA)构成



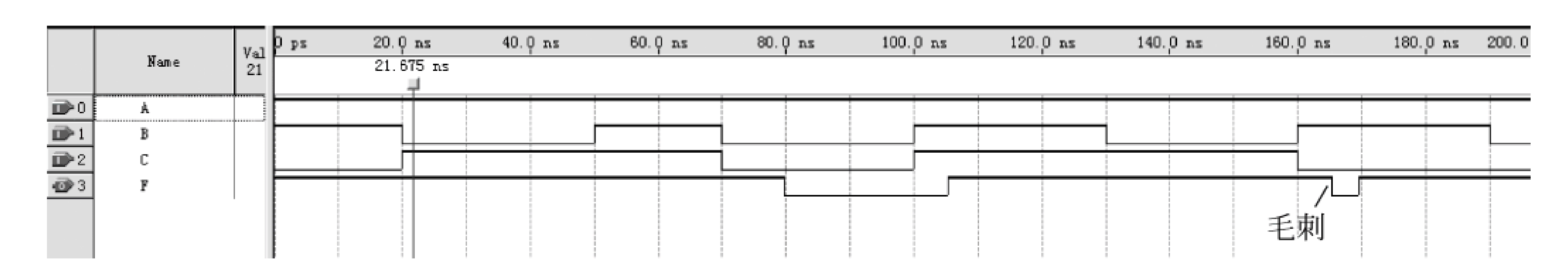
图附-3　时序仿真时产生的冒险现象

的。IOE 分布于 FPGA 芯片内部的四周，为内部逻辑阵列块和芯片外部引脚之间提供一个可编程的接口；LAB 位于芯片内部，以行列形式遍布整个芯片，用于完成用户指定的逻辑功能；PIA 位于芯片内部的逻辑块之间，用于在芯片内部各单元之间进行信号传递。由此可见，对于不同的输入，其到达同一个电路的走线是不同的，这就造成了输入信号的不同步，在输出端就有可能产生冒险现象，即产生毛刺。参见如图附-4 所示电路。



图附-4　$F=AB+AC$ 电路图

图附-4 中，$F=AB+AC$。理论上，当 A 恒为 1 时，B 和 C 中只要有一个为 1，则输出 F 就为 1。但是，由于 B 和 C 是经过不同的路径到达或门，它们到达或门的时间有可能不同步，这样就有可能产生毛刺，图附-5 是该电路在 Quartus Ⅱ 中的时序仿真波形。



图附-5　$F=AB+AC$ 电路时序仿真结果图

这种由两个以上的输入变量同时改变而产生的冒险现象称为函数冒险。

在组合逻辑电路中，一般冒险仅仅出现在信号发生变化的时候，电路瞬间会恢复稳态值，不会改变电路的性能。但是在时序逻辑电路中，冒险有可能导致电路产生错误的结果或者发生振荡。

附.2　毛刺的消除方法

避免函数冒险最简单的方法就是在同一时刻只允许有一个信号发生变化。例如，在数字电路设计中采用格雷码计数方式代替普通的二进制码计数，因为相邻的格雷码每次只有 1 位跳变，可以有效避免毛刺的产生。但是，要解决由单个参数发生改变而引起的逻辑冒险现象就要改变电路的设计。

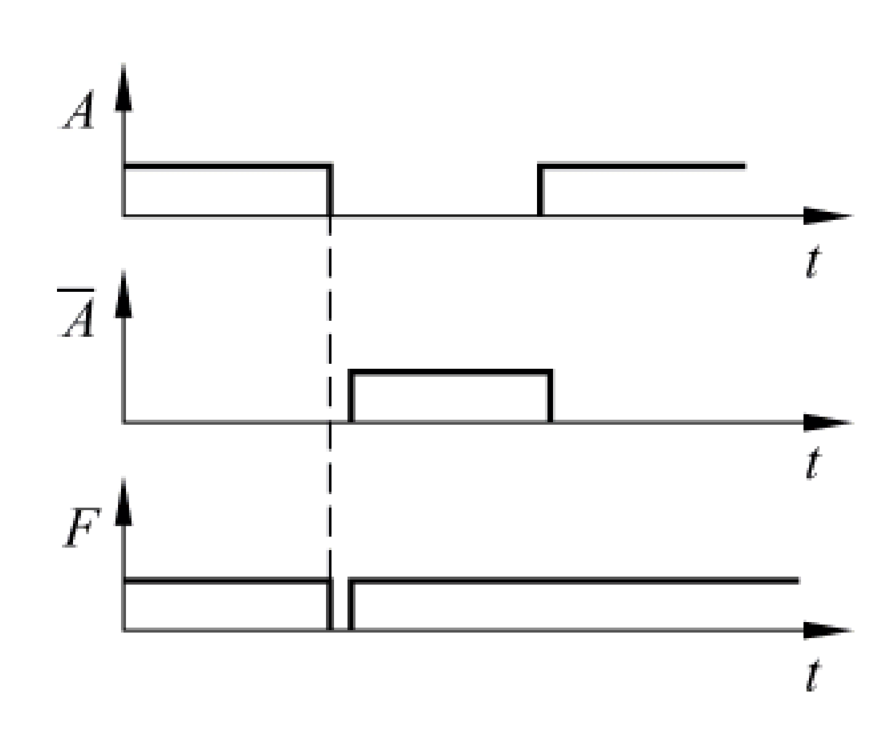
下面介绍几种简单的减少毛刺的方法。

附.2.1　利用冗余项法

电路设计过程中，经常会对电路进行化简，被化简了的电路实现起来更加简单，但是

有可能会产生毛刺，解决的方法是增加冗余项来消除毛刺。

例如，表达式 $F=AB+\overline{A}C+BC$ 可以化简为 $F=AB+\overline{A}C$，当 $B=C=1$ 时 $F=A+\overline{A}$，理论上 $F=A+\overline{A}$ 应该恒为 1。但是，当 A 由高电平向低电平转换时，A 先到达或门，值为"0"，此时或门的另外一个输入端 $\overline{A}$ 将短暂保持原来的值"0"不变，因此或门此时的输出值为"0"，在延迟一个非门（也许还有线路的延时）后新的 $\overline{A}$ 值"1"到达或门，或门的输出值恢复为"1"，如图附-6 所示。



图附-6　时序分析图

给表达式 $F=AB+\overline{A}C$ 加上冗余项 BC 变成 $F=AB+\overline{A}C+BC$ 后，即可消除毛刺：当 $B=C=1$ 时，表达式变成 $F=A+\overline{A}+1$ 不管信号 A 如何变化，都可保证 F 恒为"1"。

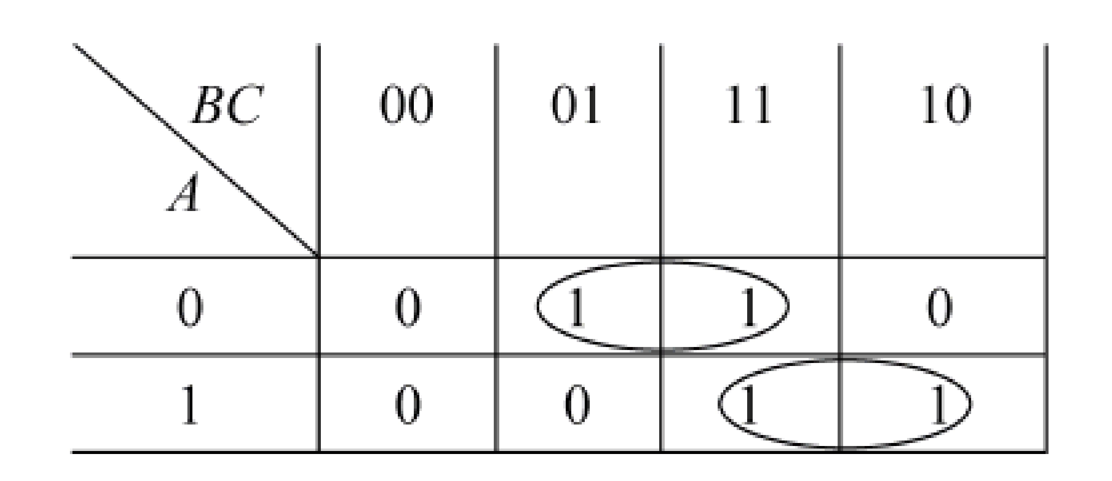
另外，在用卡诺图对真值表进行化简获得逻辑表达式时，如果化简时有两个卡诺圆相切，其对应的电路就有可能产生毛刺。仍然以 $F=AB+\overline{A}C$ 为例，其真值表如表附-1 所示。

表附-1　真值表

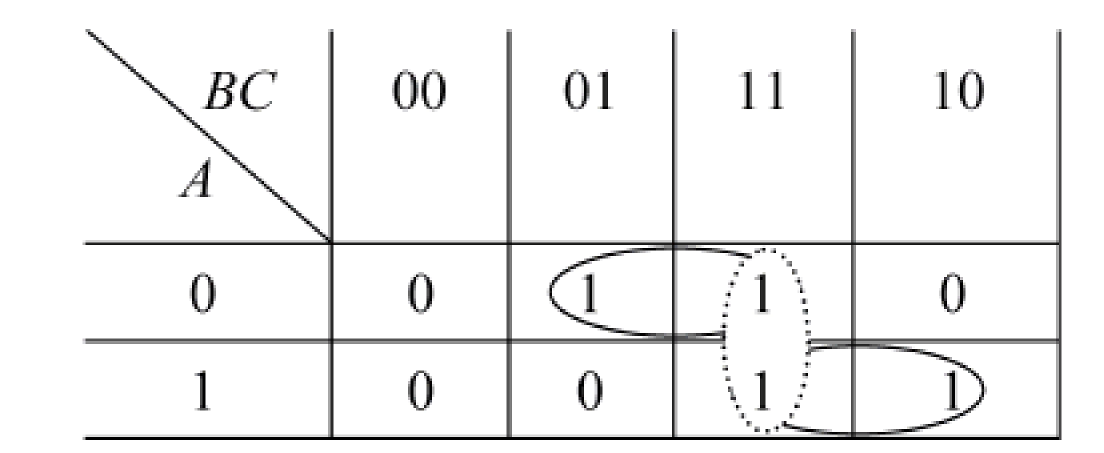
A	**B**	**C**	**F**	**A**	**B**	**C**	**F**
0	0	0	0	1	0	0	0
0	0	1	1	1	0	1	0
0	1	0	0	1	1	0	1
0	1	1	1	1	1	1	1

根据真值表画出卡诺图，进行化简，由图附-7 可见函数的逻辑表达式可化简为：$F=AB+\overline{A}C$，图中有两个卡诺圆相切，由其实现的电路就有可能产生毛刺。

解决办法是修改卡诺图，在卡诺图的两个圆相切处增加一个圆以增加冗余项来消除毛刺，如图附-8 所示，这时逻辑表达式变成 $F=AB+\overline{A}C+BC$，可以消除这一毛刺。



图附-7　卡诺图化简



图附-8　化简后增加冗余项

附.2.2　吸收法

毛刺实际上是高频窄脉冲信号，如果在产生毛刺的输出端接上一个小的滤波电容就可以滤除毛刺。但是，加上滤波电容后，输出波形的前后沿将变坏，在对波形要求比较严格的情况下不宜使用。

附.2.3　锁存法

如果毛刺发生在 D 触发器的输入端，只要毛刺不是出现在时钟的跳变沿，或者不满

足数据的建立和保持时间,毛刺就不会传送到D触发器的输出级,一般情况下,毛刺是很短的(一般为几纳秒)窄脉冲信号,基本都不满足建立和保持时间,所以说D触发器对毛刺不敏感。因此在系统的输出端加上D触发器后可有效消除毛刺。但是,系统加上了D触发器之后,会增加延时,特别是当系统级数较多的时候。因此,利用D触发器消除毛刺也要视情况而定。

附.2.4 信号延时法

由毛刺产生的根本原因可知,毛刺是由于同一个逻辑单元的两个或者两个以上的输入端到达的时间不一样,在信号改变的瞬间输出端有可能产生毛刺信号,因此,对相应信号进行延时,可以有效消除毛刺。用信号延时法消除毛刺有下列几种情况。

(1) 延迟输入信号

延迟输入信号这个方法的原理是将提前到达的信号进行延迟,使他和其他信号同时到达,就可以避免毛刺。在延时的时候要准确地估算出此信号比其他信号先到达的时间,然后确定其延时时间,这个方法实现起来比较困难。

(2) 延迟读取输出信号法

毛刺的产生一般都是在信号发生改变的时刻,过一段时间信号稳定后毛刺自动消失,因此,如果系统读取的是输出稳定后的信号就不会读到毛刺信号。延时读取输出信号法就是对下一个模块的读时钟信号进行延时,等输出信号稳定时再将其传入下一级系统。

毛刺产生的原理很简单,但是在具体电路中呈现的现象又是很复杂的,要想有效地消除毛刺,必须根据不同的电路仔细分析解决。

参 考 文 献

[1] Altera Corporation. Cyclone Ⅱ Device Family Data Sheet. July 2005，v2.0.

[2] Altera Corporation. DE2_70 User manual_v101. 2009，v1.06.

[3] Lattice Semiconductor Corporation. Specifications GAL16V8. August 2006.

[4] Altera Corporation. MAX 7000 Programmable Logic Device Family Data Sheet. September 2005. ver. 6.7.

[5] Fairchild Semiconductor Corporation. DM74LS04 Hex Inverting Gates. Revised. March 2000.

[6] Stephen Brown and Zvonko Vranesic，Fundamentals of Digital Logic with Verilog Design. China Machine Press，2007.

[7] Fairchild Semiconductor Corporation. DM74LS08 Quad 2-Input AND Gates. Revised. March 2000.

[8] Kleitz，W. 李慧军 译. VHDL 数字电子学. 北京：科学出版社，2008.

[9] 白中英. 数字逻辑与数字系统(第三版). 北京：科学出版社，2002.

[10] 张志刚. FPGA 与 SOPC 设计教程——DE2 实践. 西安：西安电子科技大学出版社，2007.

[11] Clifford E. Cummings Nonblocking Assignments in Verilog Synthesis，Coding Styles That Kill! Sunburst Design，Inc.

[12] IEEE Standard Verilog Hardware Description Language 2001，IEEE Std. 1364-2001.

[13] FPGA/SOPC 入门级实验指导书(第 2.86 版). 南京大学计算机系整理改编. 2009.9.

[14] J. Bhasker. 夏宇闻，甘伟译. Verilog HDL 入门(第 3 版). 北京：北京航空航天大学出版社，2008.

[15] http://www.asic-world.com/verilog/verilog_one_day1.html#Introduction.

[16] http://www.verilog.com/.

[17] http://www.eet-china.com/SEARCH/ART/EDA.HTM. Verilog In One Day.

[18] http://www.altera.com.cn/products/devices/cyclone2/features/io_capabilities/cy2-io_capabilities.html#single.

[19] http://www.altera.com.cn/products/devices/cyclone2/features/cy2-ext_mem_int.html.

[20] John F. Wakerly. 林生等译. 数字设计原理与实践. 北京：机械工业出版社，2008.



普通高等教育"十一五"国家级规划教材
21世纪大学本科计算机专业系列教材

近期出版书目

- 计算概论(第 2 版)
- 计算概论——程序设计阅读题解
- 计算机导论(第 2 版)
- 计算机导论教学指导与习题解答
- 计算机伦理学
- 程序设计导引及在线实践
- 程序设计基础(第 2 版)
- 程序设计基础习题解析与实验指导
- 程序设计基础(C 语言)
- 程序设计基础(C 语言)实验指导
- 离散数学(第 2 版)
- 离散数学习题解答与学习指导(第 2 版)
- 数据结构(STL 框架)
- 算法设计与分析
- 算法设计与分析(第 2 版)
- 算法设计与分析习题解答(第 2 版)
- C ++ 程序设计(第 2 版)
- Java 程序设计
- 面向对象程序设计(第 2 版)
- 形式语言与自动机理论(第 3 版)
- 形式语言与自动机理论教学参考书(第 3 版)
- 数字电子技术基础
- 数字逻辑
- FPGA 数字逻辑设计
- 计算机组成原理(第 3 版)
- 计算机组成原理教师用书(第 3 版)
- 计算机组成原理学习指导与习题解析(第 3 版)
- 微机原理与接口技术
- 微型计算机系统与接口(第 2 版)
- 计算机组成与系统结构
- 计算机组成与体系结构习题解答与教学指导
- 计算机组成与体系结构(第 2 版)
- 计算机系统结构教程
- 计算机系统结构学习指导与题解
- 计算机操作系统(第 2 版)
- 计算机操作系统学习指导与习题解答
- 编译原理
- 软件工程
- 计算机图形学
- 计算机网络(第 3 版)
- 计算机网络教师用书(第 3 版)
- 计算机网络实验指导书(第 3 版)
- 计算机网络习题解析与同步练习
- 计算机网络软件编程指导书
- 人工智能
- 多媒体技术原理及应用(第 2 版)
- 计算机网络工程(第 2 版)
- 计算机网络工程实验教程
- 信息安全原理及应用